Business Case Study Yulu – Hypothesis Testing

Name : Tanmoy Basuli

Gmail : basuli575tanmoy@gmail.com

Introduction

In this study-case, I'll give an Exploratory Data Analysis of the Yulu dataset. I will explore the data and hopefully bring some insights.

We will try to find some insights on below,

- Bi-Variate Analysis
- 2-sample t-test: testing for difference across populations
- ANNOVA
- Chi-square

We will see above analysis using Histogram, Q-Q plot or statistical methods like Levene's test, Shapiro-wilk test.

For visualizations I used, seaborn, pyplot and plotly. Some of the visuals interactive, and some of it static. But there's a lot improve. Feedback is welcome.

Yulu

Yulu is India's leading micro-mobility service provider, which offers unique vehicles for the daily commute. Starting off as a mission to eliminate traffic congestion in India, Yulu provides the safest commute solution through a user-friendly mobile app to enable shared, solo and sustainable commuting.
Yulu zones are located at all the appropriate locations (including metro stations, bus stands, office spaces, residential areas, corporate offices, etc) to make those first and last miles smooth, affordable, and convenient!
Yulu has recently suffered considerable dips in its revenues. They have contracted a consulting company to understand the factors on which the demand for these shared electric cycles depends. Specifically, they want to understand the factors affecting the demand for these shared electric cycles in the Indian market.

Business Problem

The company wants to know:

- Which variables are significant in predicting the demand for shared electric cycles in the Indian market?
- How well those variables describe the electric cycle demands

Dataset: https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/001/428/original/bike_sharing.csv?1642089089

The Outline of this notebook is as follows.

1. Basic Data Exploration
    - Feature Exploration
    - Summary Statistics
2. Data Cleaning
    - Null Value Analysis
    - Checking Duplicate Values
    - Handling inconsistent or incorrect data
3. Exploratory data analysis (What is the Story Of Data)

Importing Libraries and Loading the Dataset

```
# Import Relevant Packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import datetime as dt
from scipy.stats import probplot
from scipy.stats import chi2_contingency
from scipy.stats import levene
from scipy.stats import kruskal
from scipy.stats import mannwhitneyu
from scipy.stats import shapiro
from scipy.stats import f_oneway
from statsmodels.graphics.gofplots import qqplot
```

```
# Load the dataset

df = pd.read_csv('bike_sharing.csv')
```

Basic Data Exploration

1. Feature Exploration
2. Summary Statistics

1. Feature Exploration

First, let us look at a quick peek of what the first five rows in the data has in store for us and what features we have.

```
# First five row of the dataset

df.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0	3	13	16
1	2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0	8	32	40
2	2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0	5	27	32
3	2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0	3	10	13
4	2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0	0	1	1

```
#Look what all are the columns I have

df.columns
```

```
Index(['datetime', 'season', 'holiday', 'workingday', 'weather', 'temp',
       'atemp', 'humidity', 'windspeed', 'casual', 'registered', 'count'],
      dtype='object')
```

In this dataset we have,

- **datetime**: datetime
- **season**: season (1: spring, 2: summer, 3: fall, 4: winter)
- **holiday**: whether day is a holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedule)
- **workingday**: if day is neither weekend nor holiday is 1, otherwise is 0.
- **weather**:
    - 1: Clear, Few clouds, partly cloudy, partly cloudy
    - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
    - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
    - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- **temp**: temperature in Celsius
- **atemp**: feeling temperature in Celsius
- **humidity**: humidity
- **windspeed**: wind speed
- **casual**: count of casual users

- **registered**: count of registered users
- **count**: count of total rental bikes including both casual and registered.

Now, Let's look What types of data we have:

```
# Types of data we have

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   datetime    10886 non-null  object 
 1   season      10886 non-null  int64  
 2   holiday     10886 non-null  int64  
 3   workingday  10886 non-null  int64  
 4   weather     10886 non-null  int64  
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64  
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64  
 10  registered  10886 non-null  int64  
 11  count       10886 non-null  int64  
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

Observation (Data Type)

1. Datatype of datetime is object.
2. Datatype of season int64.
3. Datatype of holiday int64.
4. Datatype of workingday int64.
5. Datatype of weather int64.
6. Datatype of temp float64.
7. Datatype of atemp float64.
8. Datatype of humidity int64.
9. Datatype of windspeed float64.
10. Datatype of casual int64.
11. Datatype of registered int64.

12. Datatype of count object.\

How many data we have in our dataset?

```
# Checking the shape of the data

df.shape
```

(10886, 12)

Now will check is their any duplicate data we have or not. If there, we need to work on this.

```
# Check duplicate data present or not

df.isnull().values.any()
```

False

Here we can see no duplicate data present at this dataset.

```
df.isna().sum()
```

```
datetime      0
season        0
holiday       0
workingday    0
weather       0
temp          0
atemp         0
humidity      0
windspeed     0
casual        0
registered    0
count         0
dtype: int64
```

Converting the datatype of datetime column from object to datetime.

```
# Now we need to chage the datatype of datetime : object to datetype

df['datetime'] = pd.to_datetime(df['datetime'])
```

What is the time period for which the data is given ?

```
# Last date of this data

df['datetime'].max()
```

Timestamp('2012-12-19 23:00:00')

```
# First date of this data

df['datetime'].min()
```

Timestamp('2011-01-01 00:00:00')

```
# Difference of first day and last day of this dataset

df['datetime'].max() - df['datetime'].min()
```

Timedelta('718 days 23:00:00')

Creating new columns as day, month and hour for further visualization.

```
# creating a new column as day, month and hour from the datetime

df['day'] = df['datetime'].dt.day_name()
df['month'] = df['datetime'].dt.month
df['hour'] = df['datetime'].dt.hour
```

Setting up ‘datetime’ as index

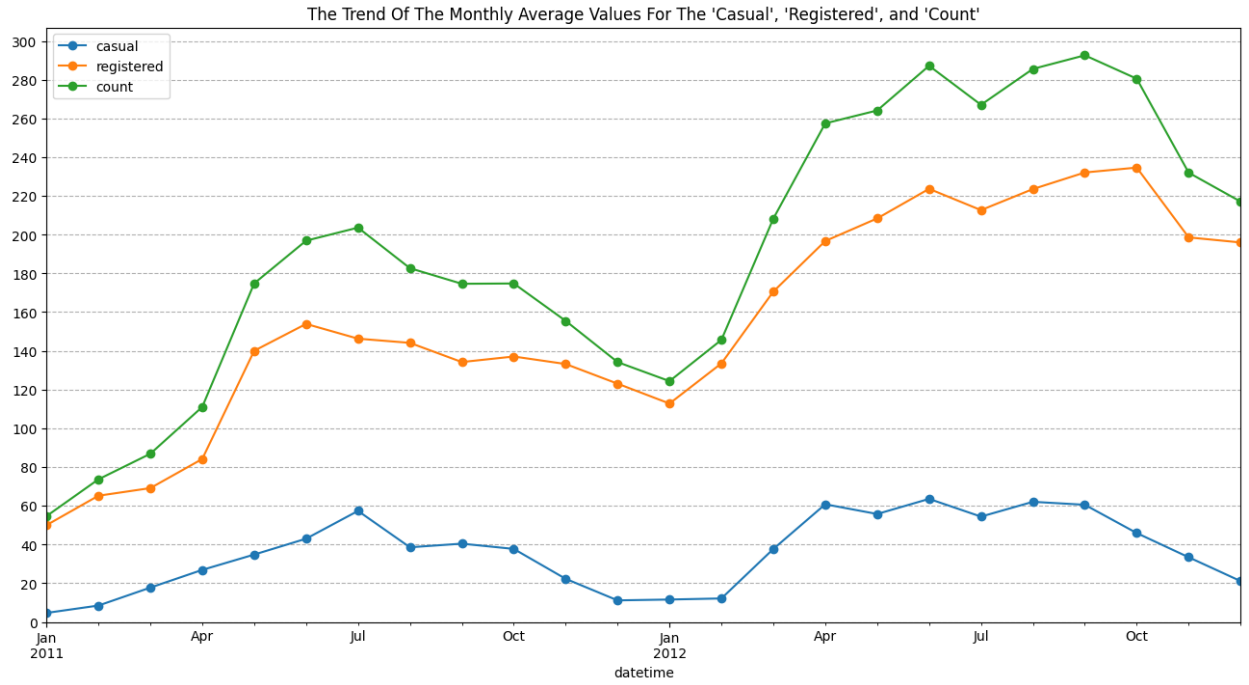
```
# setting the 'datetime' column as the index of the DataFrame 'df'

df.set_index('datetime', inplace = True)
```

By setting the 'datetime' column as the index, it allows for easier and more efficient access, filtering, and manipulation of the data based on the datetime values.

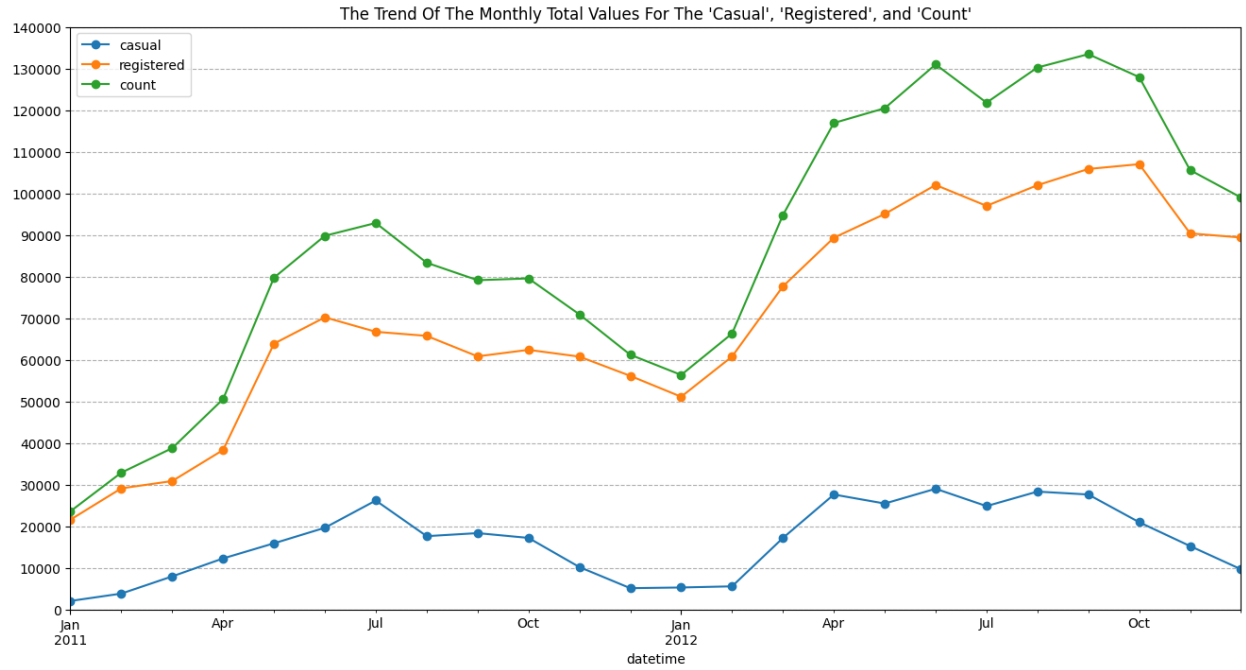
It enables operations such as resampling, slicing by specific time periods, and applying time-based calculations.

Analyzing Data by Time

```
# The below code visualizes the trend of the monthly average values for
the 'casual', 'registered',
# and 'count' variables,  allowing for easy comparison and analysis of
their patterns over time

plt.figure(figsize = (16, 8))

# plotting a lineplot by resampling the data on a monthly basis, and
calculating the mean value
    # of 'casual', 'registered' and 'count' users for each month
df.resample('M')['casual'].mean().plot(kind = 'line', legend = 'casual',
marker = 'o')
df.resample('M')['registered'].mean().plot(kind = 'line', legend =
'registered', marker = 'o')
df.resample('M')['count'].mean().plot(kind = 'line', legend = 'count',
marker = 'o')

plt.grid(axis = 'y', linestyle = '--')    # adding gridlines only along
the y-axis
plt.yticks(np.arange(0, 301, 20))
plt.ylim(0,)    # setting the lower y-axis limit to 0
plt.title("The Trend Of The Monthly Average Values For The 'Casual',
'Registered', and 'Count'")
plt.show()      # displaying the plot
```




```
# The below code visualizes the trend of the monthly total values for the
'casual', 'registered',
# and 'count' variables,  allowing for easy comparison and analysis of
their patterns over time

plt.figure(figsize = (16, 8))

# plotting a lineplot by resampling the data on a monthly basis, and
calculating the mean value
    # of 'casual', 'registered' and 'count' users for each month
df.resample('M')['casual'].sum().plot(kind = 'line', legend = 'casual',
marker = 'o')
df.resample('M')['registered'].sum().plot(kind = 'line', legend =
'registered', marker = 'o')
df.resample('M')['count'].sum().plot(kind = 'line', legend = 'count',
marker = 'o')

plt.grid(axis = 'y', linestyle = '--')    # adding gridlines only along
the y-axis
plt.yticks(np.arange(0, 140001, 10000))

plt.ylim(0,)    # setting the lower y-axis limit to 0
plt.title("The Trend Of The Monthly Total Values For The 'Casual',
'Registered', and 'Count'")
plt.show()      # displaying the plot
```



We want to know if there is an increase in the average hourly count of rental bikes from the year 2011 to 2012

```
# creating year wise avarage count of rental bikes
df1 = df.resample('Y')['count'].mean().to_frame().reset_index()

# creating a new column and shifting the value
df1['prev_count'] = df1['count'].shift(1)

# calculating percantage
df1['diff_percentage'] = (df1['count'] -
df1['prev_count'])/df1['prev_count'] * 100
df1
```

	datetime	count	prev_count	diff_percentage
0	2011-12-31	144.223349	NaN	NaN
1	2012-12-31	238.560944	144.223349	65.410764

Observation

1. This data suggests that there was positive growth in the count of the variable over the course of one year.

2. The mean total hourly count of rental bikes is 144 for the year 2011 and 239 for the year 2012.
3. An annual growth rate of 65.41 % can be seen in the demand of electric vehicles on an hourly basis.

How does the average hourly count of rental bikes vary for different month?

```
# creating month wise average rented bike
df2 = df.groupby('month')['count'].mean().round(2).reset_index()

# creating a new column of his previous
df2['prev_count'] = df2['count'].shift(1)

# creating the difeerence and measuring the percentage
df2['difference_percentage'] = ((df2['count'] -
df2['prev_count'])/df2['prev_count'] * 100).round(2)
df2
```

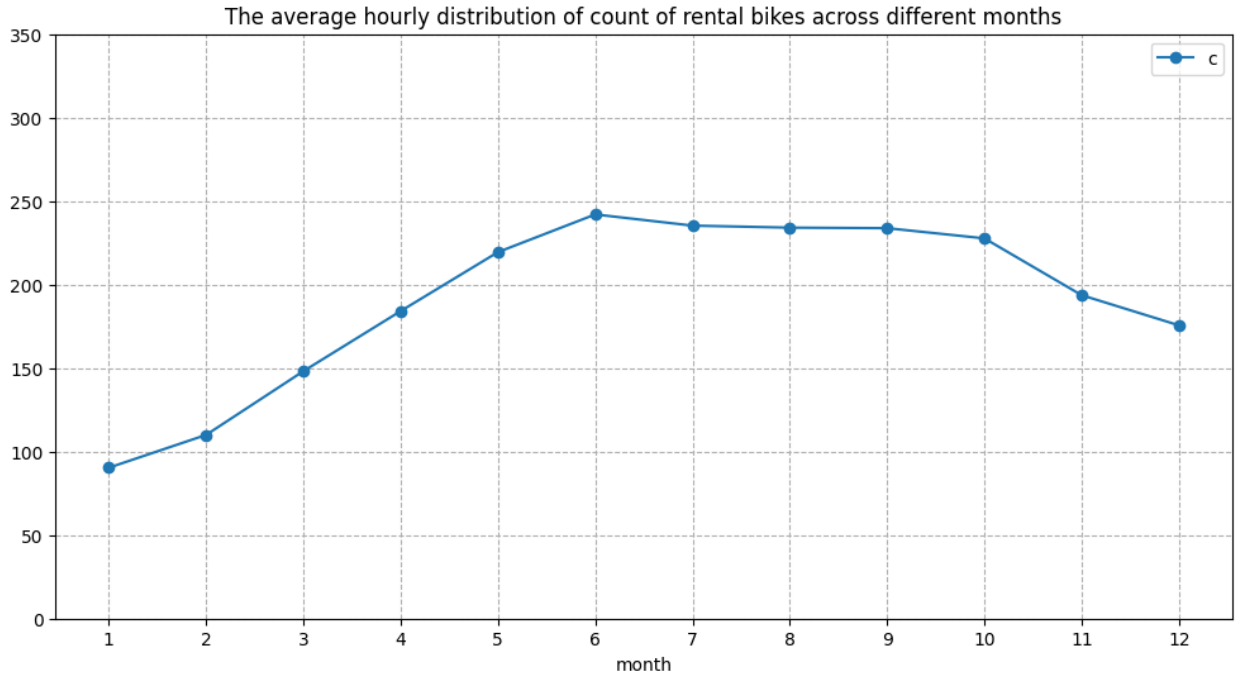
month	count	prev_count	difference_percentage
1	90.37	NaN	NaN
2	110.00	90.37	21.72
3	148.17	110.00	34.70
4	184.16	148.17	24.29
5	219.46	184.16	19.17
6	242.03	219.46	10.28
7	235.33	242.03	-2.77
8	234.12	235.33	-0.51
9	233.81	234.12	-0.13
10	227.70	233.81	-2.61
11	193.68	227.70	-14.94
12	175.61	193.68	-9.33

Observation

1. The count of rental bikes shows an increasing trend from January to March, with a significant growth rate of 34.70% between February and March.
2. The growth rate starts to stabilize from April to June, with a relatively smaller growth rate.
3. From July to September, there is a slight decrease in the count of rental bikes, with negative growth rates.
4. The count further declines from October to December, with the largest drop observed between October and November (-14.94%).

Now will visualize the data and try to find out some insights based on that.

```
# The resulting plot visualizes the average hourly distribution of the
count of rental bikes for each
# month- and allowing for comparison and identification of any patterns or
trends throughout the year.

# Setting the figure size for the plot
plt.figure(figsize = (12, 6))

# Setting the title for the plot
plt.title("The average hourly distribution of count of rental bikes across
different months")

# Grouping the DataFrame by the month and calculating the mean of the
'count' column for each month.
    # Ploting the line graph using markers ('o') to represent the average
count per month.
df.groupby('month')['count'].mean().plot(kind = 'line', marker = 'o')

plt.ylim(0,)    # Setting the y-axis limits to start from zero
plt.xticks(np.arange(1, 13))   # Setting the x-ticks to represent the
months from 1 to 12
plt.legend('count')    # Adding a legend to the plot for the 'count' line.
plt.yticks(np.arange(0, 400, 50))
# Adding gridlines to both the x and y axes with a dashed line style
plt.grid(axis = 'both', linestyle = '--')
plt.plot()     # Displaing the plot
```



Observation

1. The average hourly count of rental bikes is the highest in the month of June followed by July and August.
2. The average hourly count of rental bikes is the lowest in the month of January followed by February and March.

Overall, these trends suggest a seasonal pattern in the count of rental bikes, with higher demand during the spring and summer months, a slight decline in the fall, and a further decrease in the winter months. It could be useful for the rental bike company to consider these patterns for resource allocation, marketing strategies, and operational planning throughout the year.

What is the distribution of average count of rental bikes on an hourly basis in a single day?

```
# creating month wise average rented bike
df3 = df.groupby('hour')['count'].mean().round(2).reset_index()

# creating a new column of his previous
df3['prev_count'] = df3['count'].shift(1)

# creating the difeerence and measuring the percentage
```

```
df3['difference_percentage'] = ((df3['count'] -
df3['prev_count'])/df3['prev_count'] * 100).round(2)
df3
```

hour	count	prev_count	difference_percentage
0	55.14	NaN	NaN
1	33.86	55.14	-38.59
2	22.90	33.86	-32.37
3	11.76	22.90	-48.65
4	6.41	11.76	-45.49
5	19.77	6.41	208.42
6	76.26	19.77	285.74
7	213.12	76.26	179.46
8	362.77	213.12	70.22
9	221.78	362.77	-38.86
10	175.09	221.78	-21.05
11	210.67	175.09	20.32
12	256.51	210.67	21.76
13	257.79	256.51	0.50

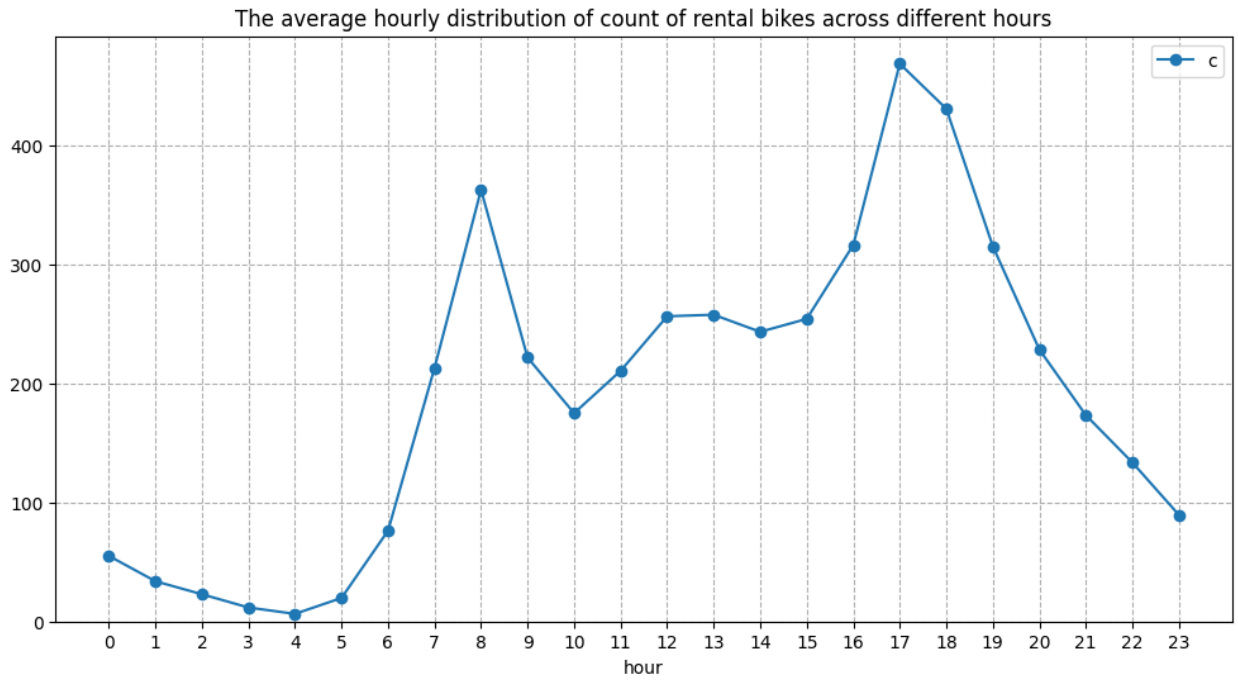
14	243.44	257.79	-5.57
15	254.30	243.44	4.46
16	316.37	254.30	24.41
17	468.77	316.37	48.17
18	430.86	468.77	-8.09
19	315.28	430.86	-26.83
20	228.52	315.28	-27.52
21	173.37	228.52	-24.13
22	133.58	173.37	-22.95
23	89.51	133.58	-32.99

Observation

1. During the early morning hours (hours 0 to 5), there is a significant decrease in the count, with negative growth percentages ranging from -38.59% to -48.66%.
2. However, starting from hour 5, there is a sudden increase in count, with a sharp positive growth percentage of 208.52% observed from hour 4 to hour 5.
3. The count continues to rise significantly until reaching its peak at hour 17, with a growth percentage of 48.17% compared to the previous hour.
4. After hour 17, there is a gradual decrease in count, with negative growth percentages ranging from -8.08% to -32.99% during the late evening and nighttime hours.

```
# The resulting plot visualizes the average hourly distribution of the
count of rental bikes for each
# hour- and allowing for comparison and identification of any patterns or
trends throughout the year.

# Setting the figure size for the plot
plt.figure(figsize = (12, 6))

# Setting the title for the plot
plt.title("The average hourly distribution of count of rental bikes across
different hours")

# Grouping the DataFrame by the month and calculating the mean of the
'count' column for each month.
```

```
    # Ploting the line graph using markers ('o') to represent the average
count per month.
df.groupby('hour')['count'].mean().plot(kind = 'line', marker = 'o')

plt.ylim(0,)    # Setting the y-axis limits to start from zero
plt.xticks(np.arange(1, 13))   # Setting the x-ticks to represent the
months from 1 to 12
plt.legend('count')    # Adding a legend to the plot for the 'count' line.
plt.yticks(np.arange(0, 400, 50))
# Adding gridlines to both the x and y axes with a dashed line style
plt.grid(axis = 'both', linestyle = '--')
plt.plot()     # Displaing the plot
```



Observation

1. The average count of rental bikes is the highest at 5 PM followed by 6 PM and 8 AM of the day.
2. The average count of rental bikes is the lowest at 4 AM followed by 3 AM and 5 AM of the day.

These patterns indicate that there is a distinct fluctuation in count throughout the day, with low counts during early morning hours, a sudden increase in the morning, a peak count in the afternoon, and a gradual decline in the evening and nighttime.

Season has mentioned 1,2,3 and 4. Need to categorize this data.

```
#  categorize season (1: spring, 2: summer, 3: fall, 4: winter)
def season_category_yulu(x):
  if x == 1 :
    return 'spring'
  elif x == 2 :
    return "summer"
  elif x == 3 :
    return 'fall'
  else :
    return 'winter'

# applying this to season column
df['season'] = df['season'].apply(season_category_yulu)
```

Basic Description of the dataset

```
# updating holiday column as category
df['holiday'] = df['holiday'].astype('category')

# updating workingday column as category
df['workingday'] = df['workingday'].astype('category')

# updating season column as category
df['season'] = df['season'].astype('category')

# updating weather column to category
df['weather'] = df['weather'].astype('category')
```

```
# describing the data
df.describe()
```

	datetime	temp	atemp	humidity	windspeed	casual	registered	count
count	10886	10886.00000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000	10886.000000
mean	2011-12-27 05:56:22.399411968	20.23086	23.655084	61.886460	12.799395	36.021955	155.552177	191.574132
min	2011-01-01 00:00:00	0.82000	0.760000	0.000000	0.000000	0.000000	0.000000	1.000000
25%	2011-07-02 07:15:00	13.94000	16.665000	47.000000	7.001500	4.000000	36.000000	42.000000
50%	2012-01-01 20:30:00	20.50000	24.240000	62.000000	12.998000	17.000000	118.000000	145.000000
75%	2012-07-01 12:45:00	26.24000	31.060000	77.000000	16.997900	49.000000	222.000000	284.000000
max	2012-12-19 23:00:00	41.00000	45.455000	100.000000	56.996900	367.000000	886.000000	977.000000
std	NaN	7.79159	8.474601	19.245033	8.164537	49.960477	151.039033	181.144454

Observation

These statistics provide insights into the central tendency, spread, and range of the numerical features in the dataset.

```
# checking all the season inside data
(df['season'].value_counts(normalize = True)*100).round(2)
```

```
season
winter    25.11
fall      25.11
summer    25.11
spring    24.67
Name: proportion, dtype: float64
```

```
# checking all the holiday inside data
(df['holiday'].value_counts(normalize = True)*100).round(2)
```

```
holiday
0    97.14
1     2.86
Name: proportion, dtype: float64
```

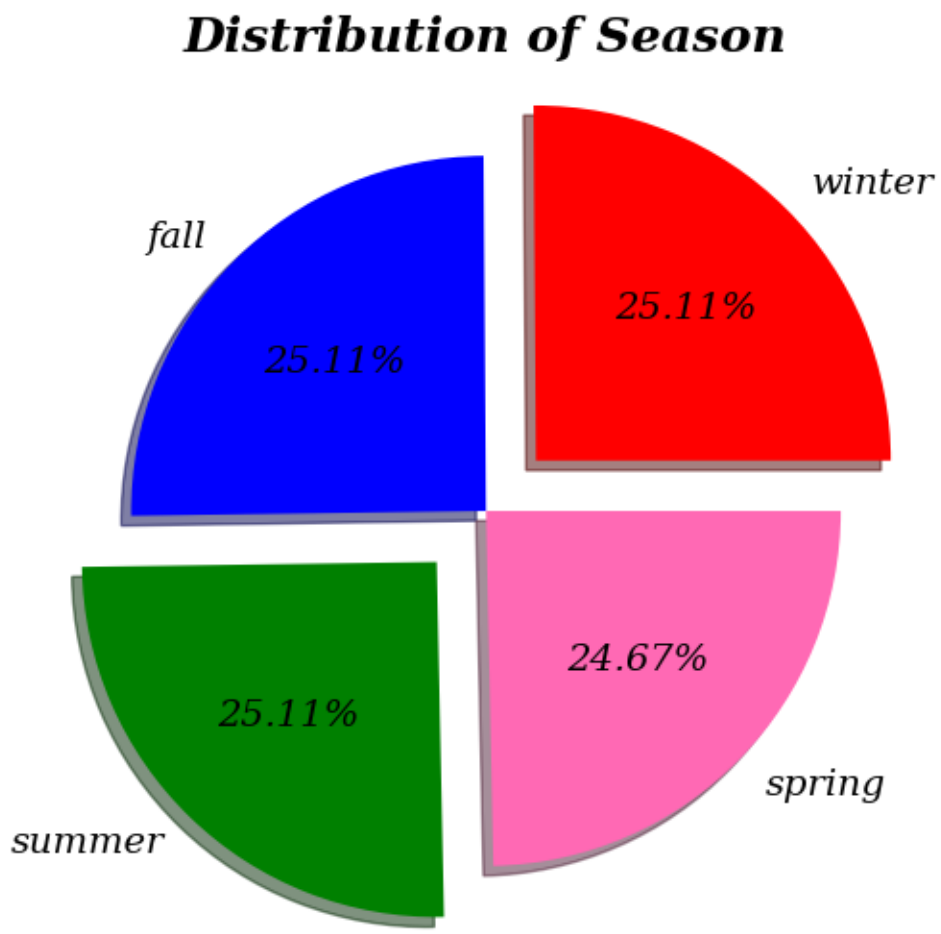
```
# checking all the workingday inside data
(df['workingday'].value_counts(normalize = True)*100).round(2)
```

```
workingday
1    68.09
0    31.91
Name: proportion, dtype: float64
```

```
# checking all the weather inside data
(df['weather'].value_counts(normalize = True)*100).round(2)
```

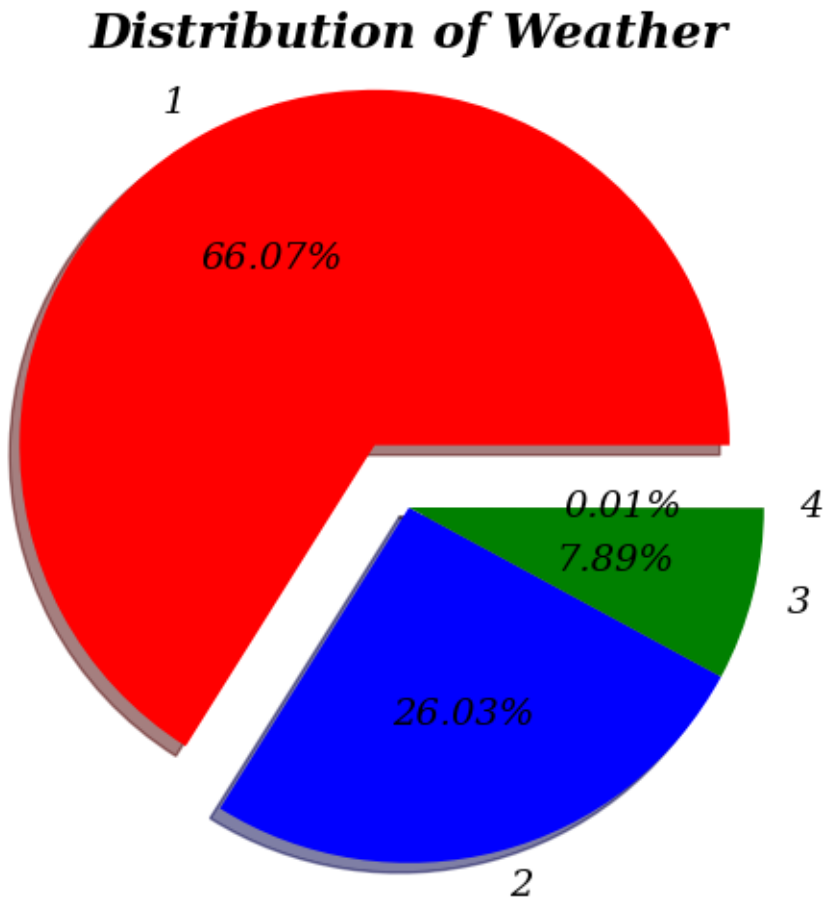
weather
1 66.07
2 26.03
3 7.89
4 0.01
Name: proportion, dtype: float64

```
# The below code helps us a visually appealing pie chart to showcase the
distribution of seasons in the dataset

# setting the figure size to 6*6
plt.figure(figsize = (6, 6))

# creating the season values
df_season = (df['season'].value_counts(normalize = True)*100).round(2)

# creating the pie-chart
plt.pie(x = df_season.values,
        labels= df_season.index,
        explode = [0.2,0.0,0.2,0.0],
        shadow = True,
        colors = ['red','blue','green','hotpink'],
        autopct = '%.2f%%',
        textprops = {'fontsize' : 14,
                     'fontstyle' : 'oblique',
                     'fontfamily' : 'serif',
                     'fontweight' : 500})
plt.title('Distribution of Season', fontdict = {'fontsize' : 18,
                                                'fontweight' : 600,
                                                'fontstyle' : 'oblique',
                                                'fontfamily' : 'serif'})
plt.show()
```



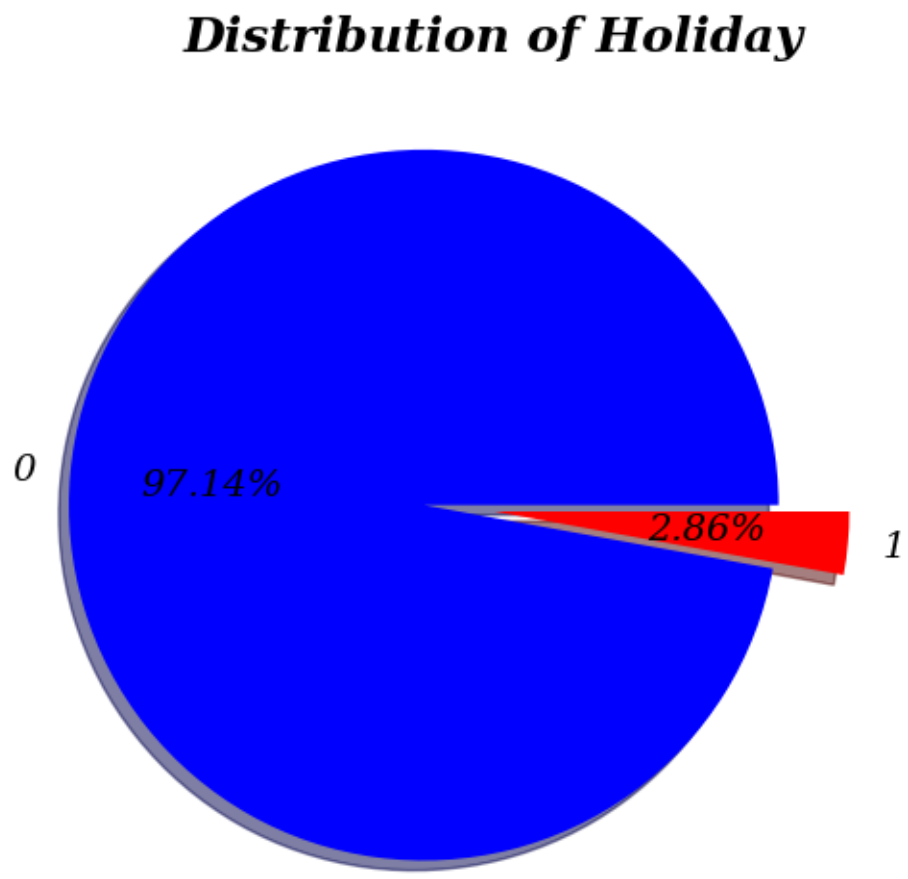

```
# The below code helps us a visually appealing pie chart to showcase the
distribution of weather in the dataset

# setting the figure size to 6*6
plt.figure(figsize = (6, 6))

# creating the weather values
df_weather = (df['weather'].value_counts(normalize = True)*100).round(2)

# creating the pie-chart
plt.pie(x = df_weather.values,
        labels= df_weather.index,
        explode = [0.2,0.0,0.0,0.0],
        shadow = True,
        colors = ['red','blue','green','hotpink'],
        autopct = '%.2f%%',
        textprops = {'fontsize' : 14,
                     'fontstyle' : 'oblique',
                     'fontfamily' : 'serif',
```

```
                    'fontweight' : 500})
plt.title('Distribution of Weather', fontdict = {'fontsize' : 18,
                                                 'fontweight' : 600,
                                                 'fontstyle' : 'oblique',
                                                 'fontfamily' : 'serif'})
plt.show()
```



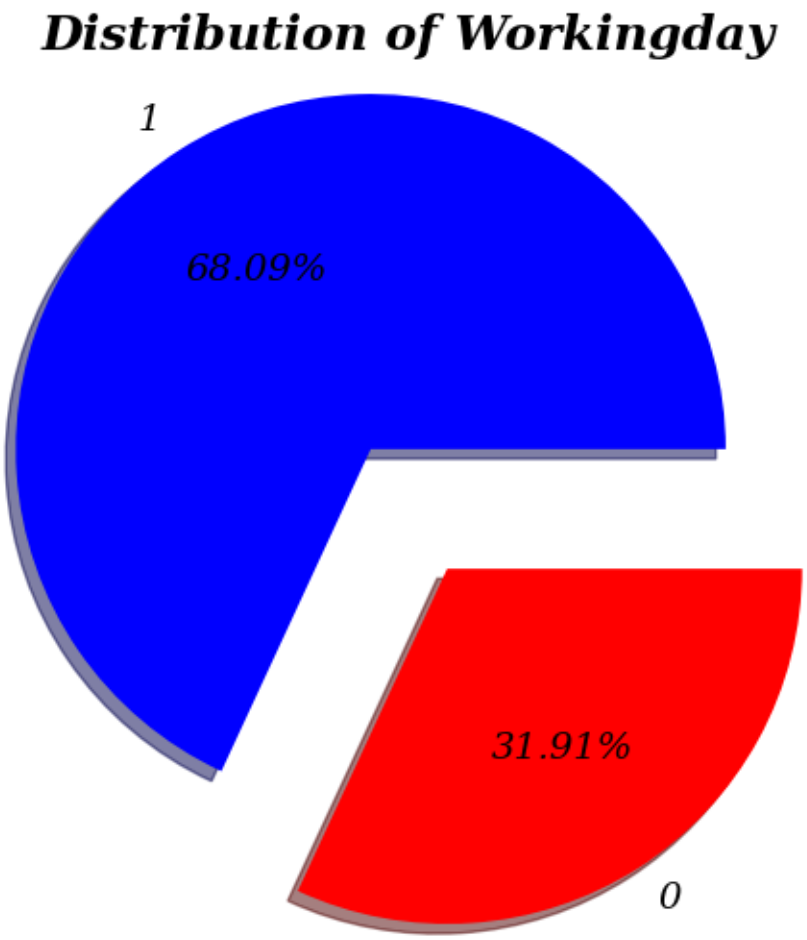

```
# The below code helps us a visually appealing pie chart to showcase the
distribution of holiday in the dataset

# setting the figure size to 6*6
plt.figure(figsize = (6, 6))

# creating the weather values
df_holiday = (df['holiday'].value_counts(normalize = True)*100).round(2)

# creating the pie-chart
plt.pie(x = df_holiday.values,
```

```
        labels= df_holiday.index,
        explode = [0.2,0.0],
        shadow = True,
        colors = ['blue','red'],
        autopct = '%.2f%%',
        textprops = {'fontsize' : 14,
                     'fontstyle' : 'oblique',
                     'fontfamily' : 'serif',
                     'fontweight' : 500})
plt.title('Distribution of Holiday', fontdict = {'fontsize' : 18,
                                                 'fontweight' : 600,
                                                 'fontstyle' : 'oblique',
                                                 'fontfamily' : 'serif'})
plt.show()
```




```
# The below code helps us a visually appealing pie chart to showcase the
distribution of workingday in the dataset
```

```
# setting the figure size to 6*6
plt.figure(figsize = (6, 6))

# creating the weather values
df_workingday = (df['workingday'].value_counts(normalize =
True)*100).round(2)

# creating the pie-chart
plt.pie(x = df_workingday.values,
        labels= df_workingday.index,
        explode = [0.2,0.2],
        shadow = True,
        colors = ['blue','red'],
        autopct = '%.2f%%',
        textprops = {'fontsize' : 14,
                     'fontstyle' : 'oblique',
                     'fontfamily' : 'serif',
                     'fontweight' : 500})
plt.title('Distribution of Workingday', fontdict = {'fontsize' : 18,
                                                    'fontweight' : 600,
                                                    'fontstyle' : 'oblique',
                                                    'fontfamily' : 'serif'})
plt.show()
```



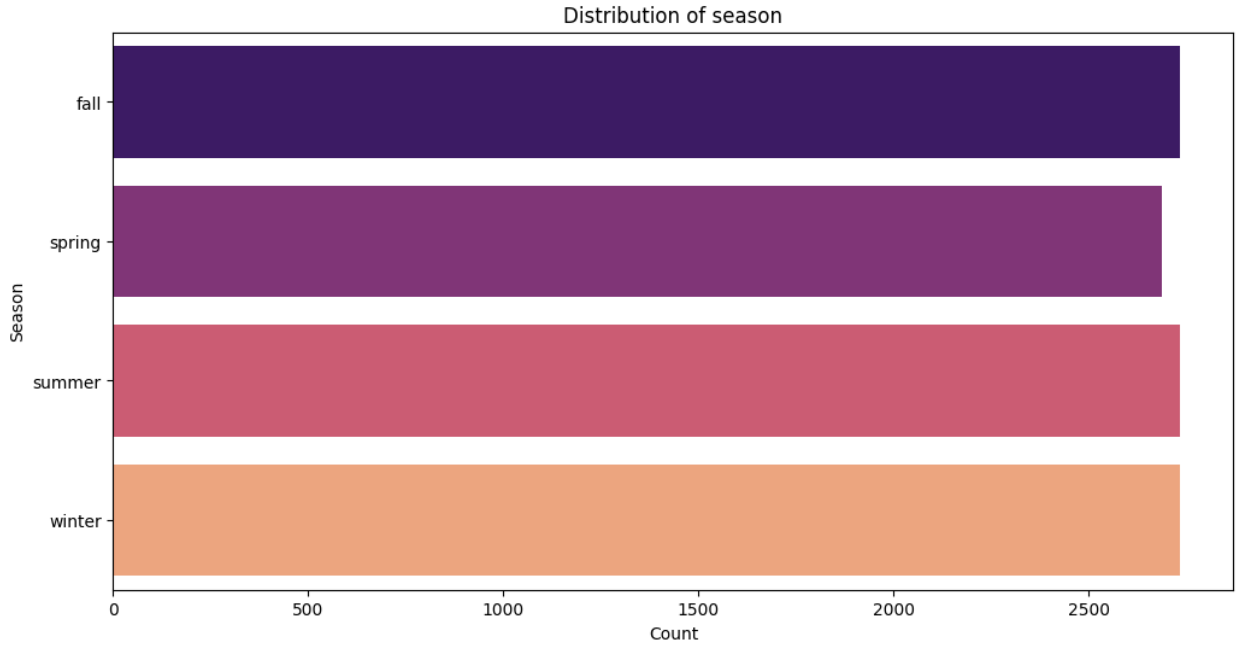
Observation

1. Seasons in Yulu are equally distributed. (winter -- 25.11%, fall -- 25.11%, summer -- 25.11%, spring -- 24.67%).
2. It is used more by customers on working day as 68%.
3. Clearly observed bikes are mostly used when weather conditions were Clear, Few clouds, partly cloudy, partly cloudy and second most is Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist.

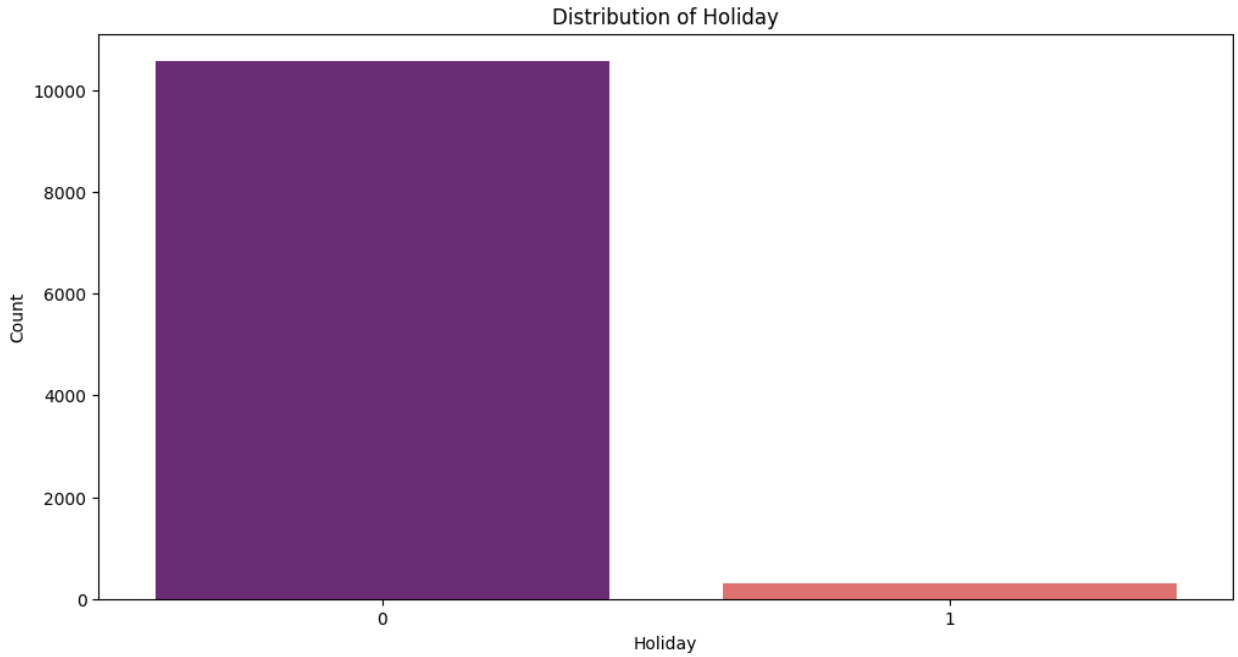
Univariate Analysis

```
#count of the season
counts_of_season = df['season'].value_counts()

# Create a bar plot
plt.figure(figsize=(12, 6))  # Set the figure size
```

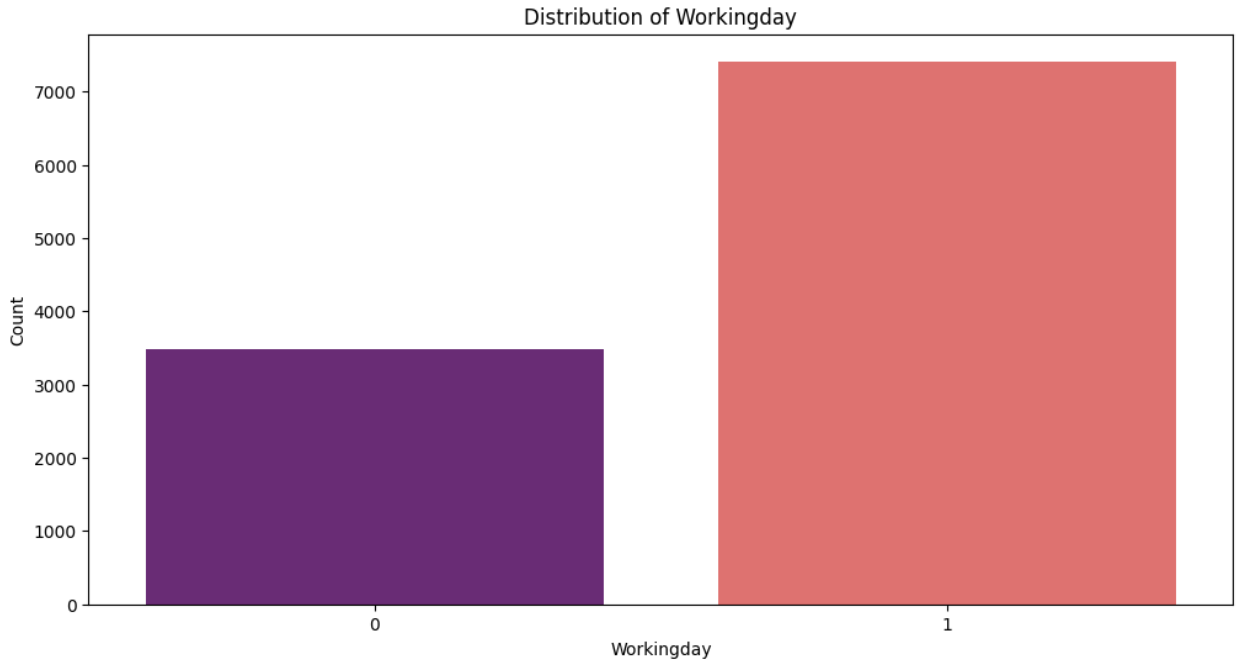
```
sns.barplot(x=counts_of_season.values, y=counts_of_season.index,
palette='magma')

# Add labels and title
plt.xlabel('Count')
plt.ylabel('Season')
plt.title('Distribution of season')

# Display the plot
plt.show()
```



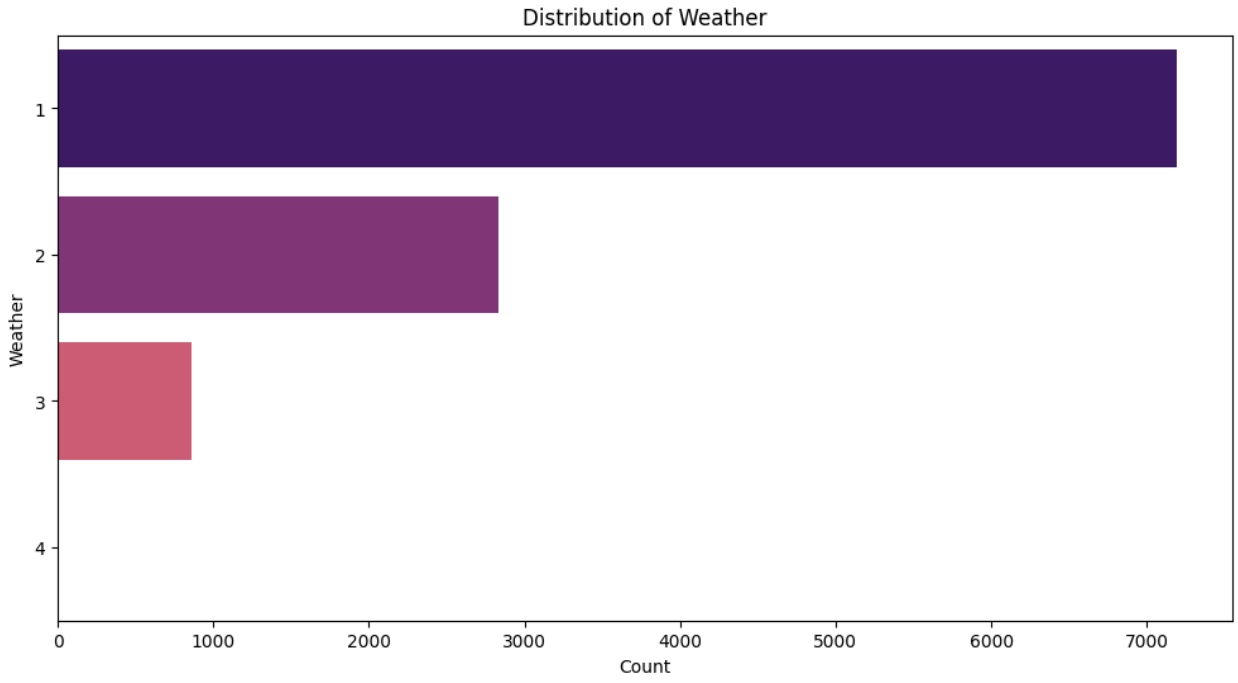

```
#count of the holiday
counts_of_holiday = df['holiday'].value_counts()

# Create a bar plot
plt.figure(figsize=(12, 6))  # Set the figure size
sns.barplot(x=counts_of_holiday.index, y=counts_of_holiday.values,
palette='magma')

# Add labels and title
plt.xlabel('Holiday')
plt.ylabel('Count')
plt.title('Distribution of Holiday')

# Display the plot
plt.show()
```



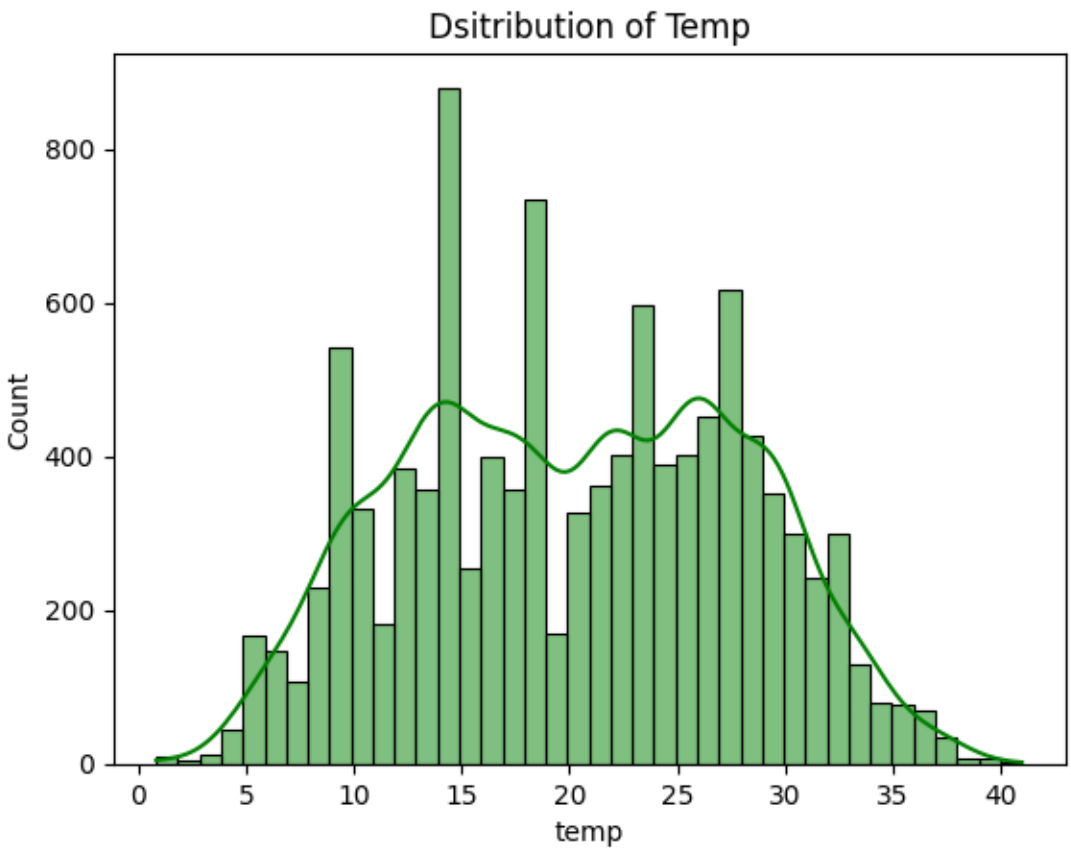

```
#count of the workingday
counts_of_workingday = df['workingday'].value_counts()

# Create a bar plot
plt.figure(figsize=(12, 6))  # Set the figure size
sns.barplot(x=counts_of_workingday.index, y=counts_of_workingday.values,
palette='magma')

# Add labels and title
plt.xlabel('Workingday')
plt.ylabel('Count')
plt.title('Distribution of Workingday')

# Display the plot
plt.show()
```




```
#count of the weather
counts_of_weather = df['weather'].value_counts()

# Create a bar plot
plt.figure(figsize=(12, 6))  # Set the figure size
sns.barplot(x=counts_of_weather.values, y=counts_of_weather.index,
palette='magma')

# Add labels and title
plt.xlabel('Count')
plt.ylabel('Weather')
plt.title('Distribution of Weather')

# Display the plot
plt.show()
```




```
# The below code generates a histogram plot for the 'temp' feature,
showing the distribution of
# temperature values in the dataset.
# The addition of the kernel density estimation plot provides a visual
representation of the underlying distribution shape, making it easier to
analyze the data distribution.

sns.histplot(data = df, x = 'temp', kde = True, bins = 40, color =
'green')
plt.title("Dsitribution of Temp")
plt.plot()
```




```
# mean and standard deviation of temp in dataset

temp_mean_yulu = df['temp'].mean().round(2)
temp_std_yulu = df['temp'].std().round(2)
temp_mean_yulu, temp_std_yulu
```

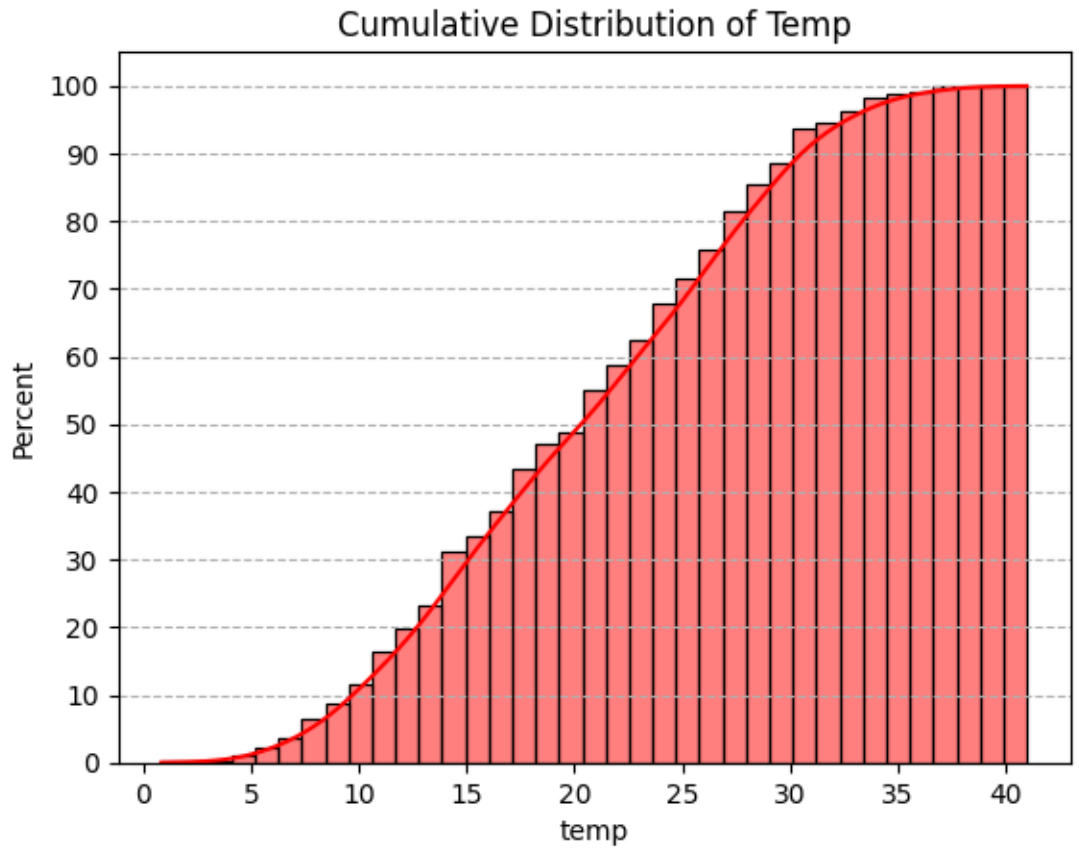
(20.23, 7.79)

Observation

1. The mean value of temperature is 20.23 where the standard deviation of temperature is 7.79 Celsius.

```
# The below code generates a histogram plot for the 'temp' feature,
showing the cumulative distribution of temperature values in the dataset.
# The addition of the kernel density estimation plot provides a visual
representation of the underlying distribution shape, making it easier to
analyze the data distribution.
```

```
sns.histplot(data = df, x = 'temp', kde = True, cumulative = True, stat =
'percent', color = 'red')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.title("Cumulative Distribution of Temp")
plt.plot()          # displaying the chart
```

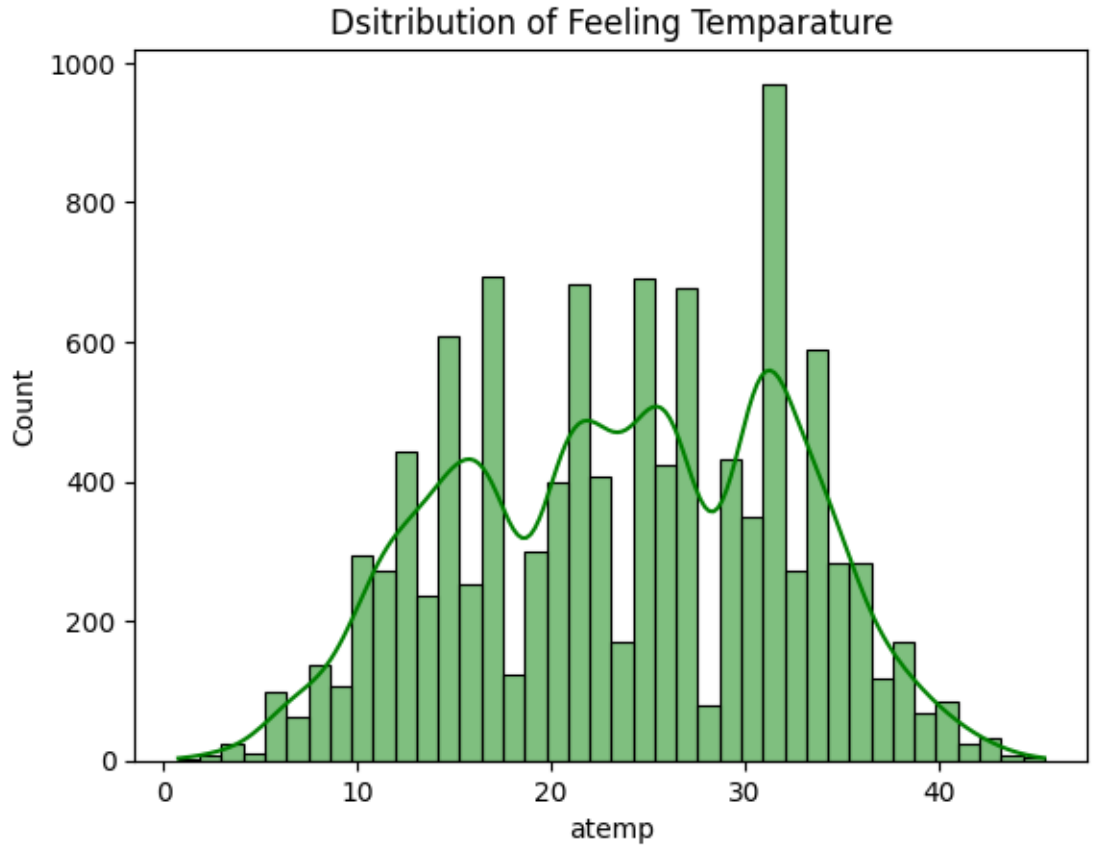


Observation

1. Can clearly observer more than 70% time temperature was less them 25 degree Celsius.

```
# The below code generates a histogram plot for the 'atemp' feature,
showing the distribution of
# temperature values in the dataset.
```

```
# The addition of the kernel density estimation plot provides a visual
representation of the underlying distribution shape, making it easier to
analyze the data distribution.

sns.histplot(data = df, x = 'atemp', kde = True, bins = 40, color =
'green')
plt.title("Dsitribution of Feeling Temparature")
plt.plot()
```



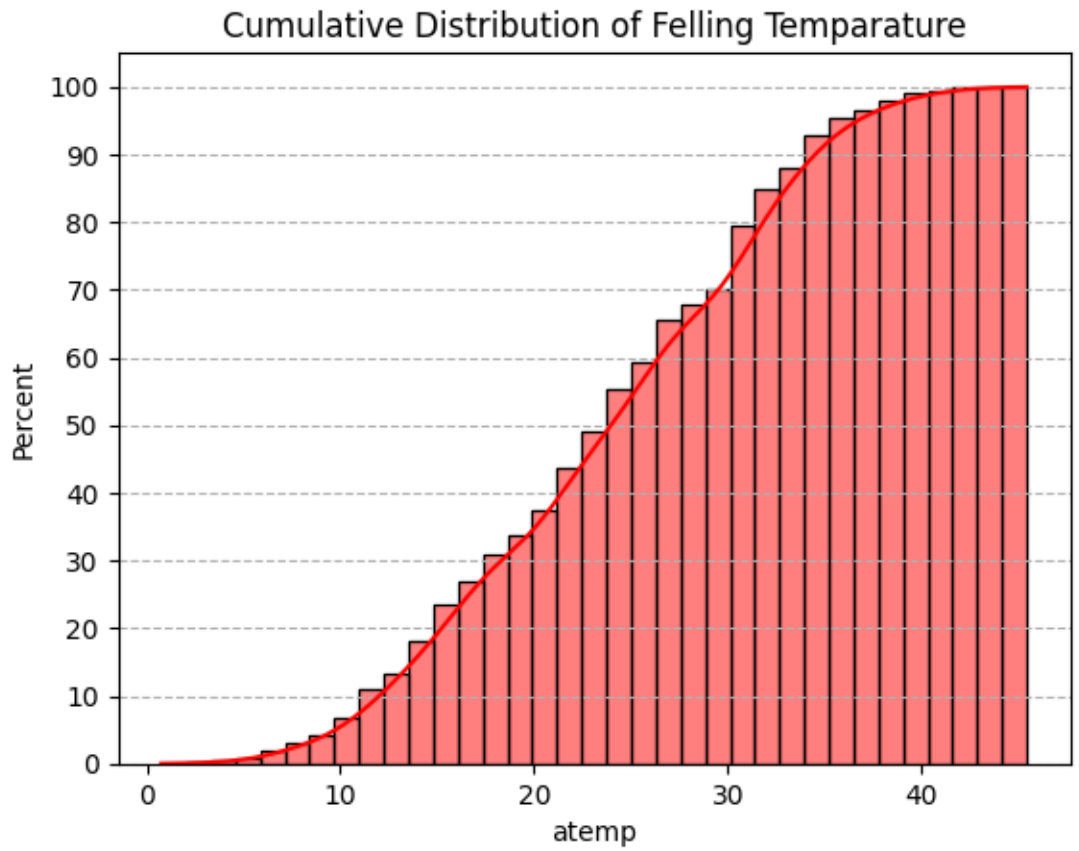

```
# mean and standard deviation of  feeling temp in dataset

temp_mean_yulu = df['atemp'].mean().round(2)
temp_std_yulu = df['atemp'].std().round(2)
temp_mean_yulu, temp_std_yulu
```

(23.66, 8.47)

Observation

1. The mean value of feeling temperature is 23.66 where the standard deviation of feeling temperature is 8.47 Celsius.

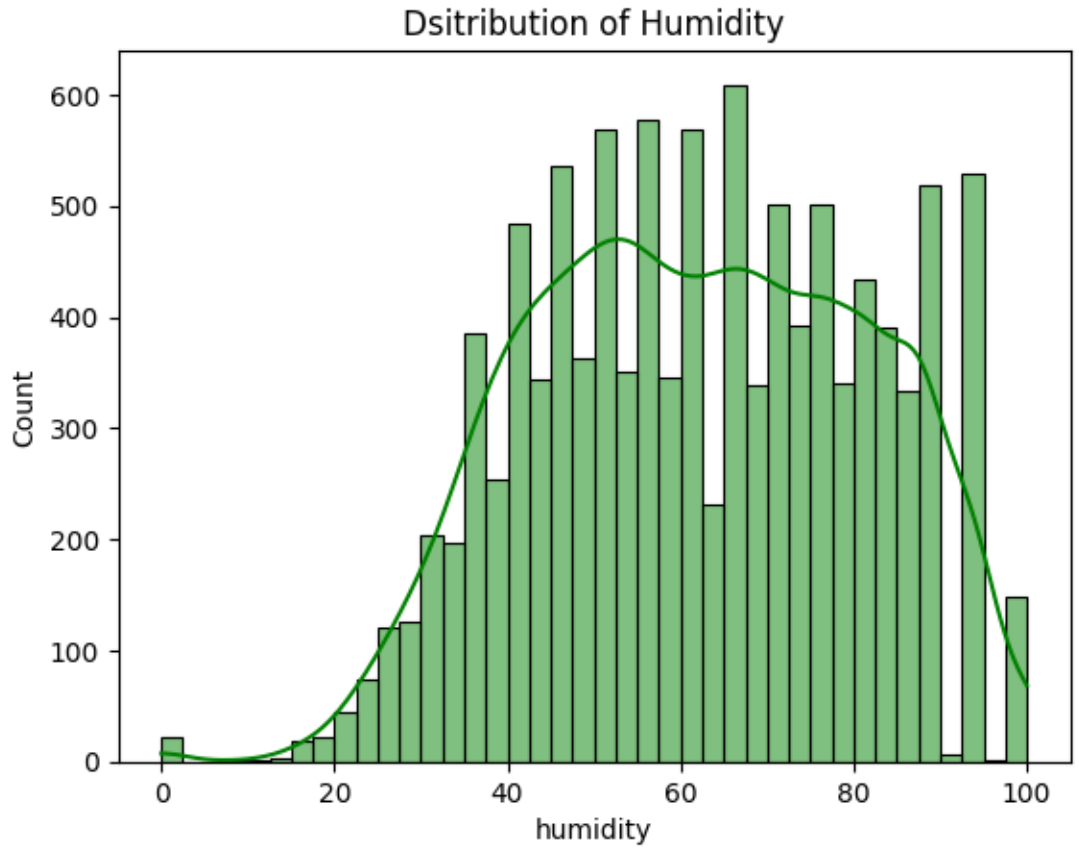
```
# The below code generates a histogram plot for the 'atemp' feature,
showing the cumulative distribution of temperature values in the dataset.
# The addition of the kernel density estimation plot provides a visual
representation of the underlying distribution shape, making it easier to
analyze the data distribution.

sns.histplot(data = df, x = 'atemp', kde = True, cumulative = True, stat =
'percent', color = 'red')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.title("Cumulative Distribution of Felling Temparature")
plt.plot()          # displaying the chart
```



Observation

1. Can clearly observer more than 70% time feeling temperature was greater than 30 degree Celsius.

```
# The below code generates a histogram plot for the 'humidity' feature, showing the distribution of
# temperature values in the dataset.
# The addition of the kernel density estimation plot provides a visual representation of the underlying distribution shape, making it easier to analyze the data distribution.

sns.histplot(data = df, x = 'humidity', kde = True, bins = 40, color = 'green')
plt.title("Dsitribution of Humidity")
plt.plot()
```




```
# mean and standard deviation of  feeling temp in dataset
```

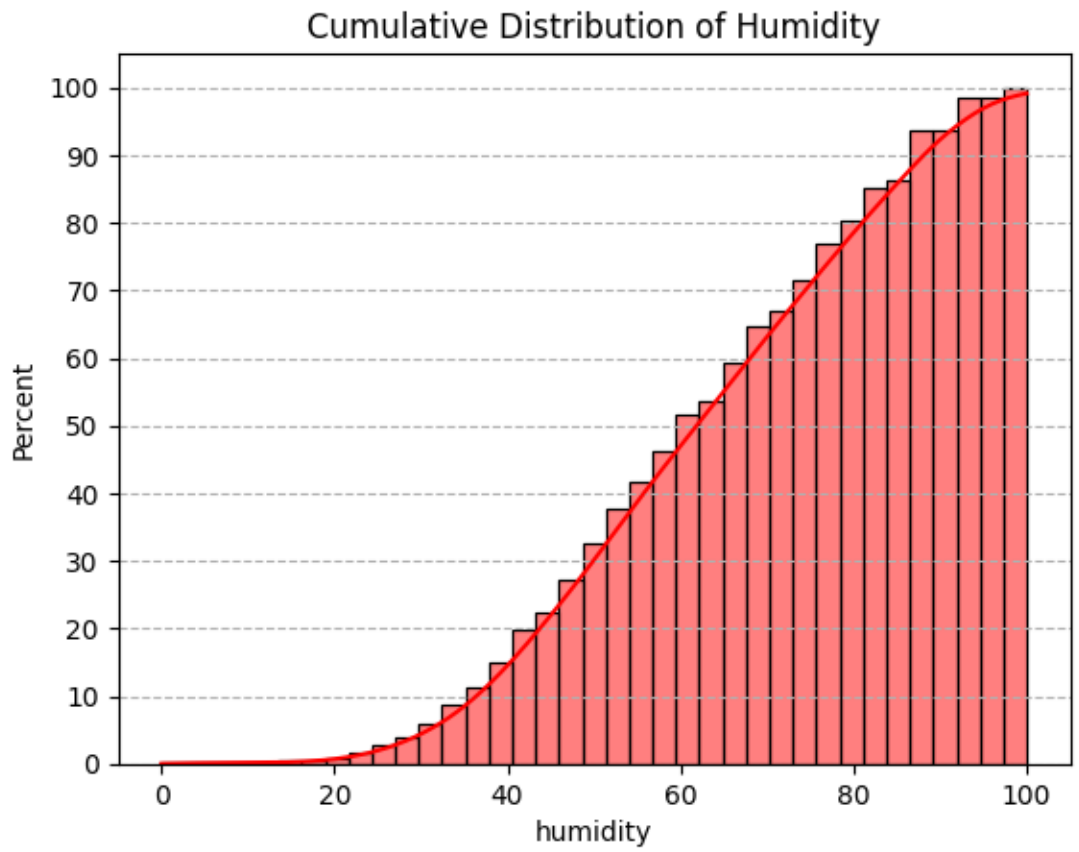
```
humidity_mean_yulu = df['humidity'].mean().round(2)
humidity_std_yulu = df['humidity'].std().round(2)
humidity_mean_yulu, humidity_std_yulu
```

(61.89, 19.25)

Observation

1. The mean value of humidity is 61.89 where the standard deviation of humidity is 19.25.

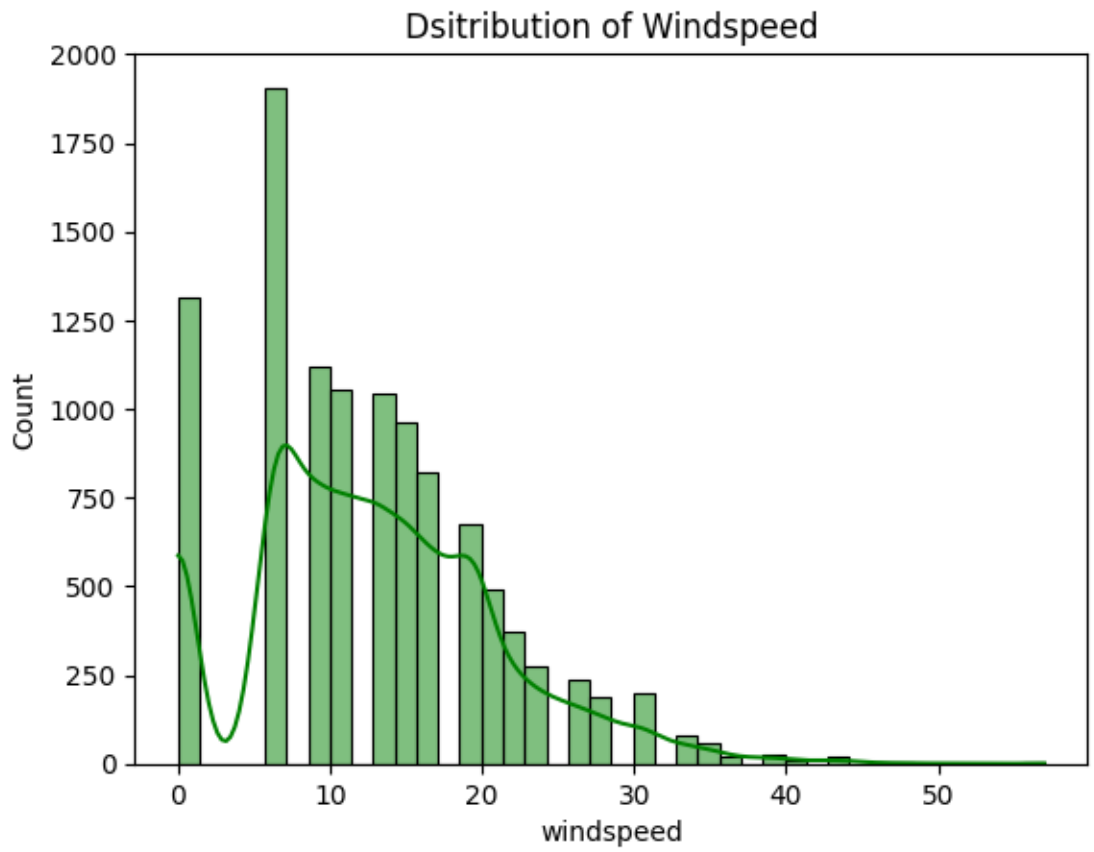
```
# The below code generates a histogram plot for the 'humidity' feature,
showing the cumulative distribution of temperature values in the dataset.
# The addition of the kernel density estimation plot provides a visual
representation of the underlying distribution shape, making it easier to
analyze the data distribution.

sns.histplot(data = df, x = 'humidity', kde = True, cumulative = True,
stat = 'percent', color = 'red')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.title("Cumulative Distribution of Humidity")
plt.plot()          # displaying the chart
```



Observation

1. Can clearly observer more than 20%-time humidity was greater than 40 degree.

```
# The below code generates a histogram plot for the 'windspeed' feature,
showing the distribution of
# temperature values in the dataset.
# The addition of the kernel density estimation plot provides a visual
representation of the underlying distribution shape, making it easier to
analyze the data distribution.

sns.histplot(data = df, x = 'windspeed', kde = True, bins = 40, color =
'green')
plt.title("Dsitribution of Windspeed")
plt.plot()
```




```
# mean and standard deviation of  feeling temp in dataset

windspeed_mean_yulu = df['windspeed'].mean().round(2)
windspeed_std_yulu = df['windspeed'].std().round(2)
windspeed_mean_yulu, windspeed_std_yulu
```

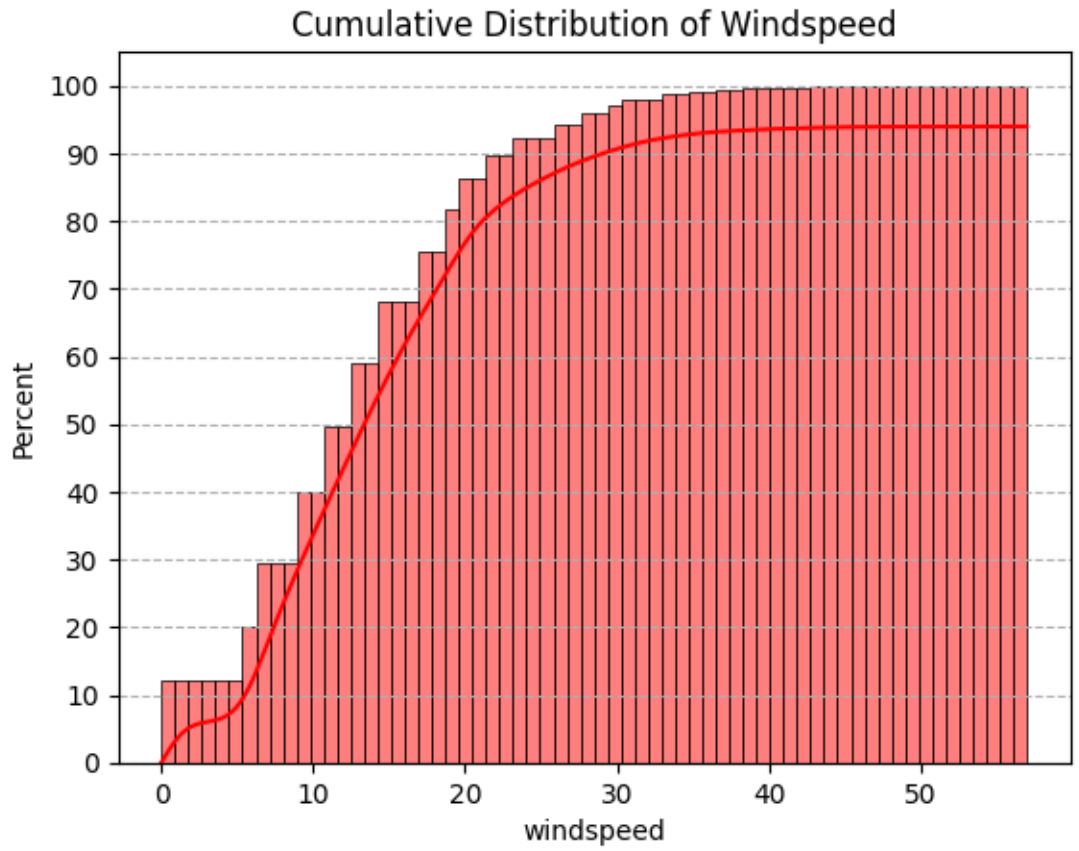
(12.8, 8.16)

Observation

1. The mean value of windspeed is 12.8 where the standard deviation of windspeed is 8.16.

```
# The below code generates a histogram plot for the 'windspeed' feature,
showing the cumulative distribution of temperature values in the dataset.
# The addition of the kernel density estimation plot provides a visual
representation of the underlying distribution shape, making it easier to
analyze the data distribution.
```

```
sns.histplot(data = df, x = 'windspeed', kde = True, cumulative = True,
stat = 'percent', color = 'red')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.title("Cumulative Distribution of Windspeed")
plt.plot()           # displaying the chart
```

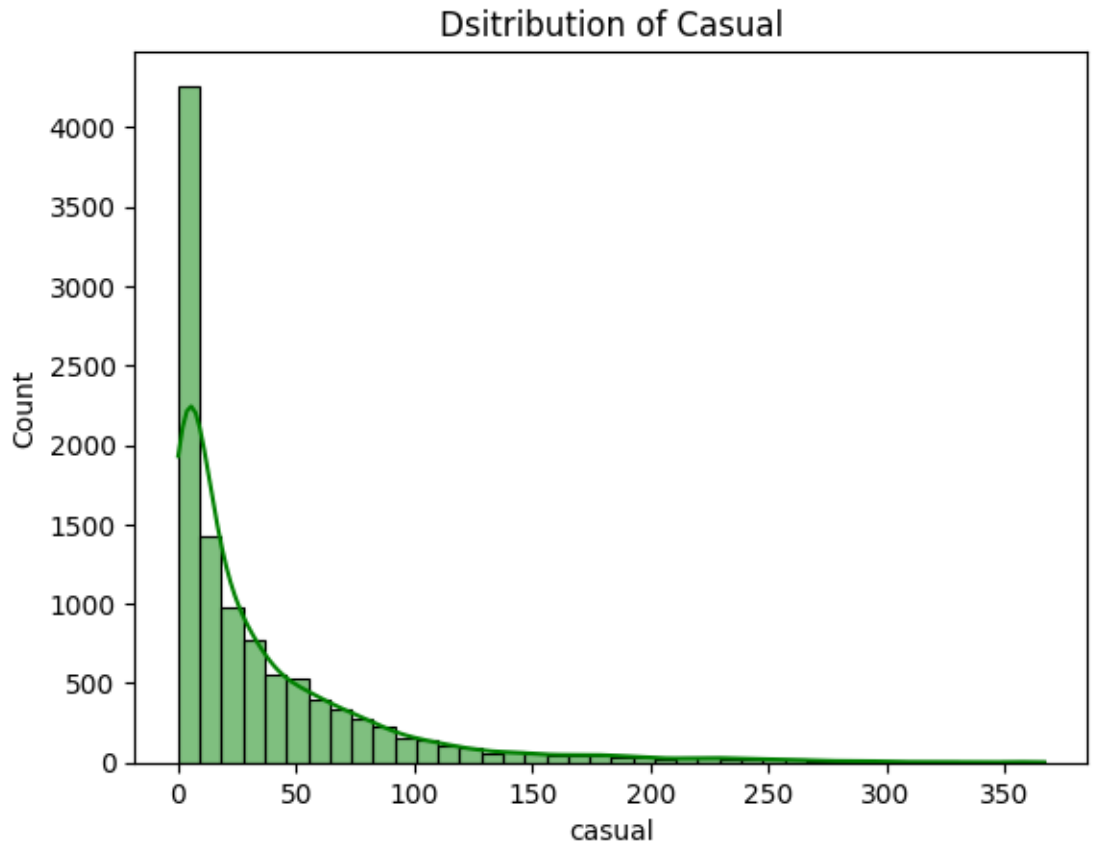


Observation

1. Can clearly observer more than 80%-time windspeed was greater than 20.

```
# The below code generates a histogram plot for the 'casual' feature,
showing the distribution of
# temperature values in the dataset.
# The addition of the kernel density estimation plot provides a visual
representation of the underlying distribution shape, making it easier to
analyze the data distribution.
```

```
sns.histplot(data = df, x = 'casual', kde = True, bins = 40, color =
'green')
plt.title("Dsitribution of Casual")
plt.plot()
```



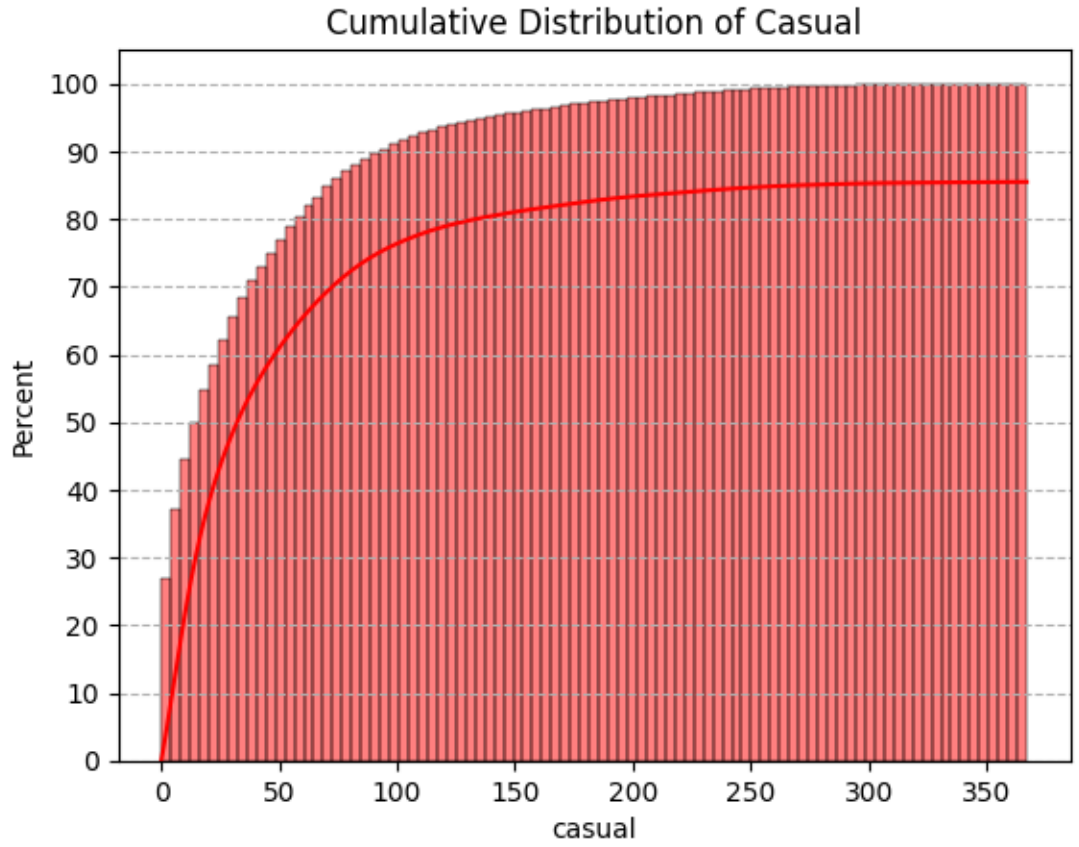

```
# mean and standard deviation of  feeling temp in dataset

casual_mean_yulu = df['casual'].mean().round(2)
casual_std_yulu = df['casual'].std().round(2)
casual_mean_yulu, casual_std_yulu
```

(36.02, 49.96)

Observation

1. The mean value of casual is 36.02 where the standard deviation of casual is 49.26.

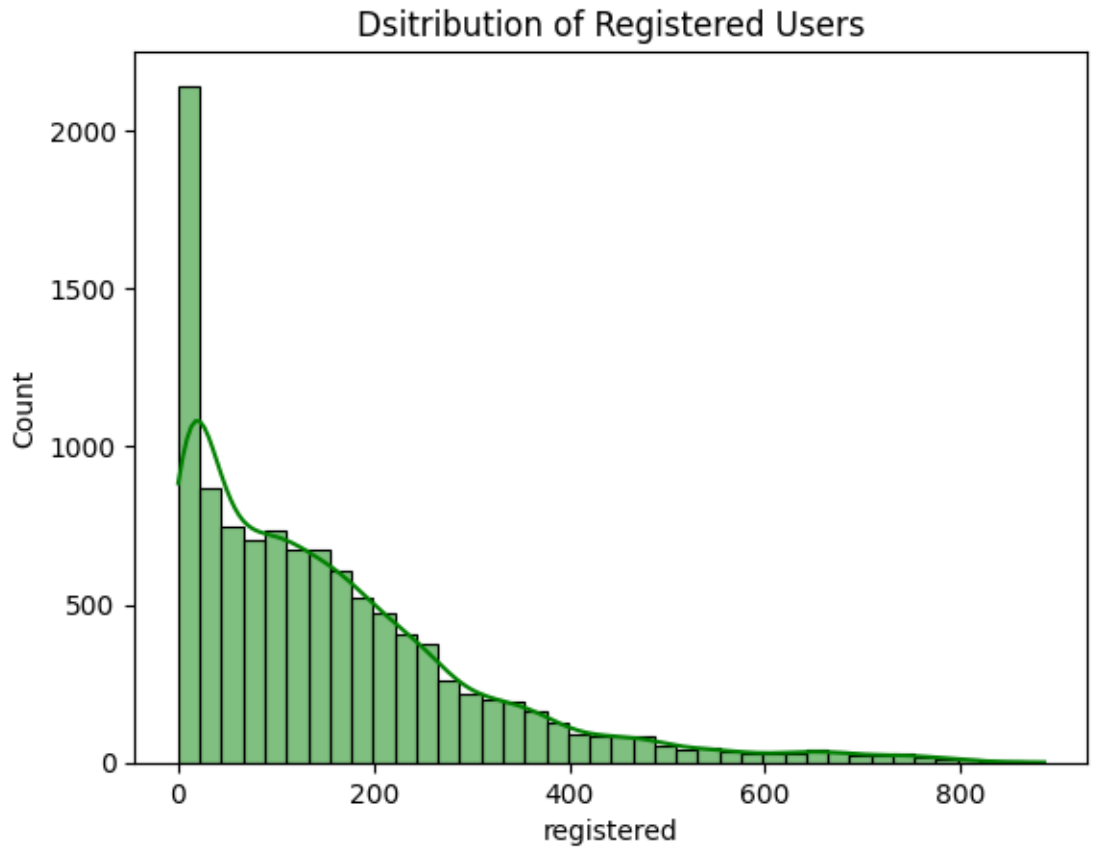
```
# The below code generates a histogram plot for the 'casual' feature,
showing the cumulative distribution of temperature values in the dataset.
# The addition of the kernel density estimation plot provides a visual
representation of the underlying distribution shape, making it easier to
analyze the data distribution.

sns.histplot(data = df, x = 'casual', kde = True, cumulative = True, stat
= 'percent', color = 'red')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.title("Cumulative Distribution of Casual")
plt.plot()           # displaying the chart
```



Observation

1. Can clearly observer more than 80%-time casual users were less than 60.

```
# The below code generates a histogram plot for the 'registered' feature,
showing the distribution of
# temperature values in the dataset.
# The addition of the kernel density estimation plot provides a visual
representation of the underlying distribution shape, making it easier to
analyze the data distribution.

sns.histplot(data = df, x = 'registered', kde = True, bins = 40, color =
'green')
plt.title("Dsitribution of Registered Users")
plt.plot()
```



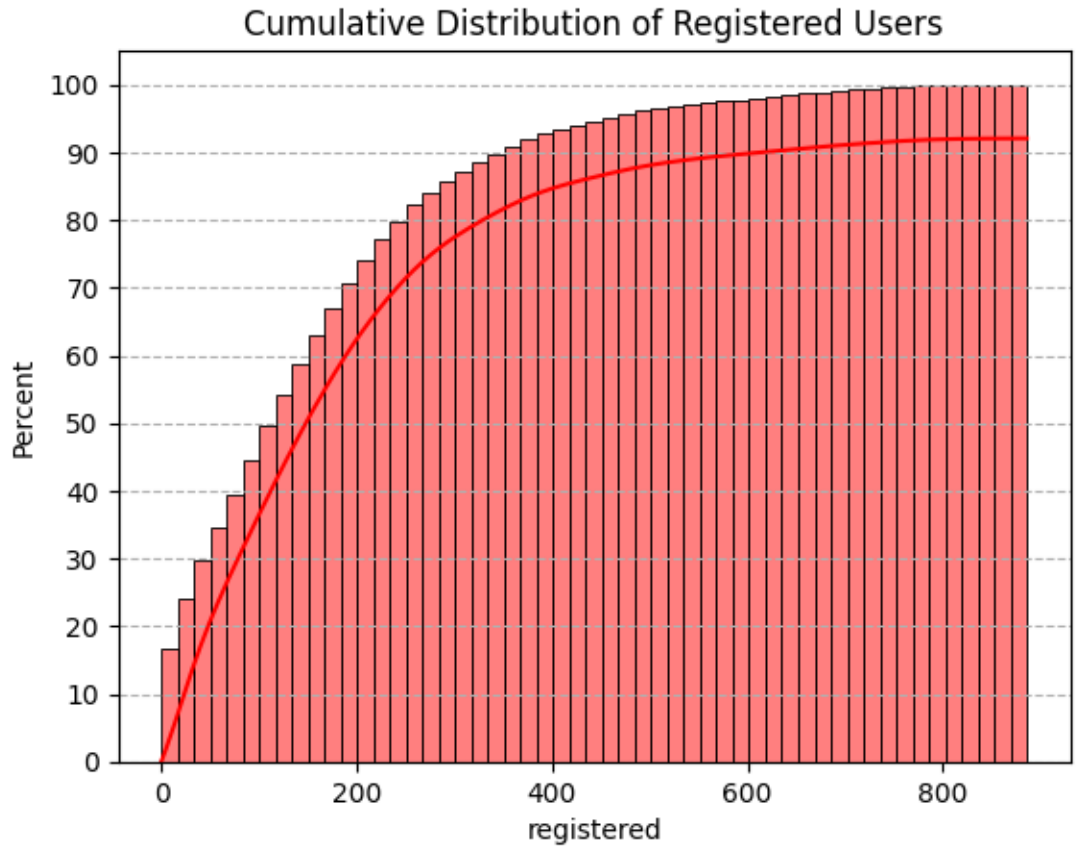

```
# mean and standard deviation of  feeling temp in dataset

registered_mean_yulu = df['registered'].mean().round(2)
registered_std_yulu = df['registered'].std().round(2)
registered_mean_yulu, registered_std_yulu
```

(155.55, 151.04)

Observation

1. The mean value of registered users is 155.55 where the standard deviation of registered users is 151.04.

```
# The below code generates a histogram plot for the 'registered' feature,
showing the cumulative distribution of temperature values in the dataset.
# The addition of the kernel density estimation plot provides a visual
representation of the underlying distribution shape, making it easier to
analyze the data distribution.

sns.histplot(data = df, x = 'registered', kde = True, cumulative = True,
stat = 'percent', color = 'red')
plt.grid(axis = 'y', linestyle = '--')
plt.yticks(np.arange(0, 101, 10))
plt.title("Cumulative Distribution of Registered Users")
plt.plot()          # displaying the chart
```

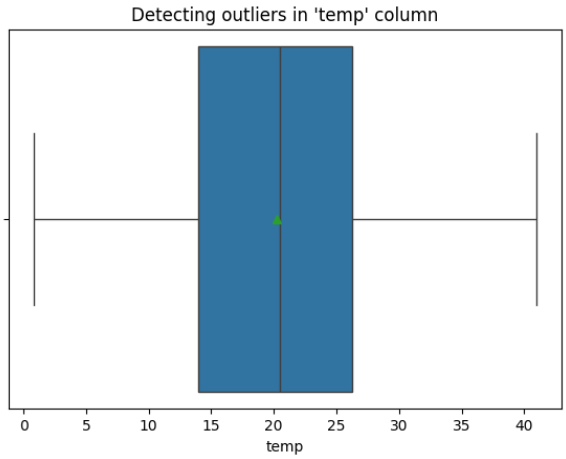


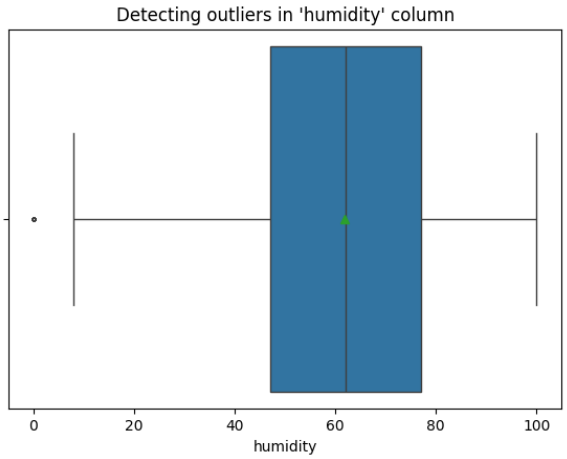
Observation

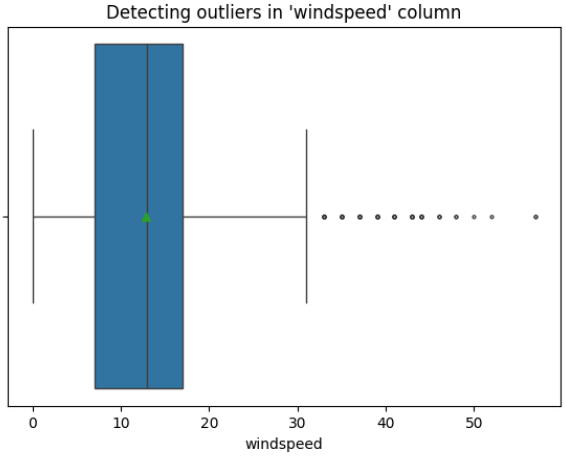
1. Can clearly observer more than 85%-time the count of registered users were less than 300.

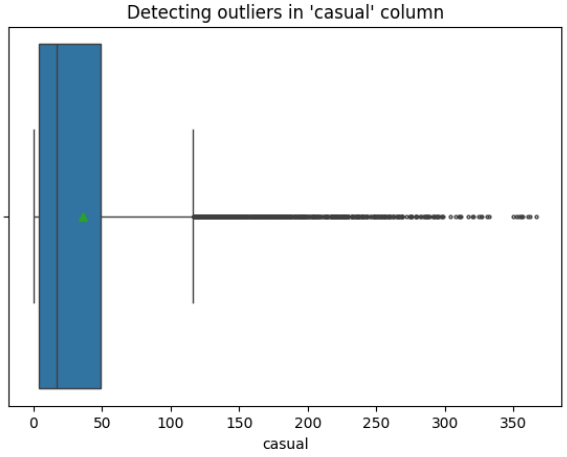
Outliers Detection

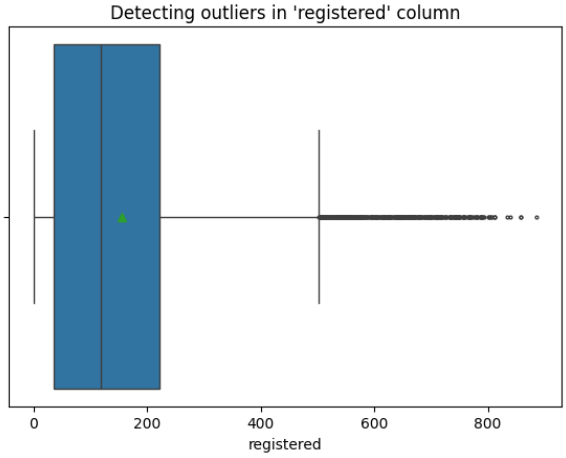
```
columns = ['temp', 'humidity', 'windspeed', 'casual', 'registered',
'count']

cp = 1
plt.figure(figsize = (15,16))
for i in columns:
    plt.subplot(3, 2, cp)
    plt.title(f"Detecting outliers in '{i}' column")
    sns.boxplot(data = df, x = df[i], showmeans = True, fliersize = 2)
    plt.plot()
    cp += 1
```

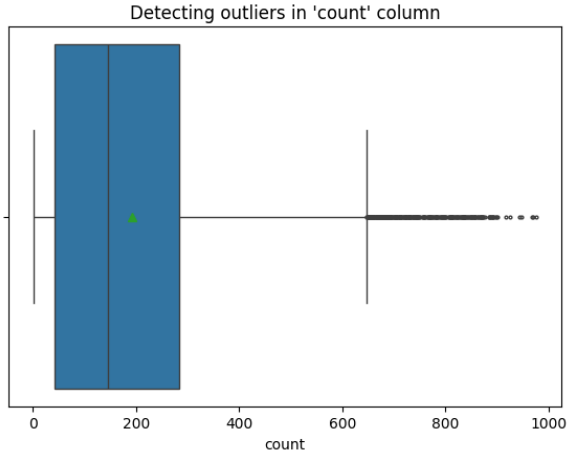










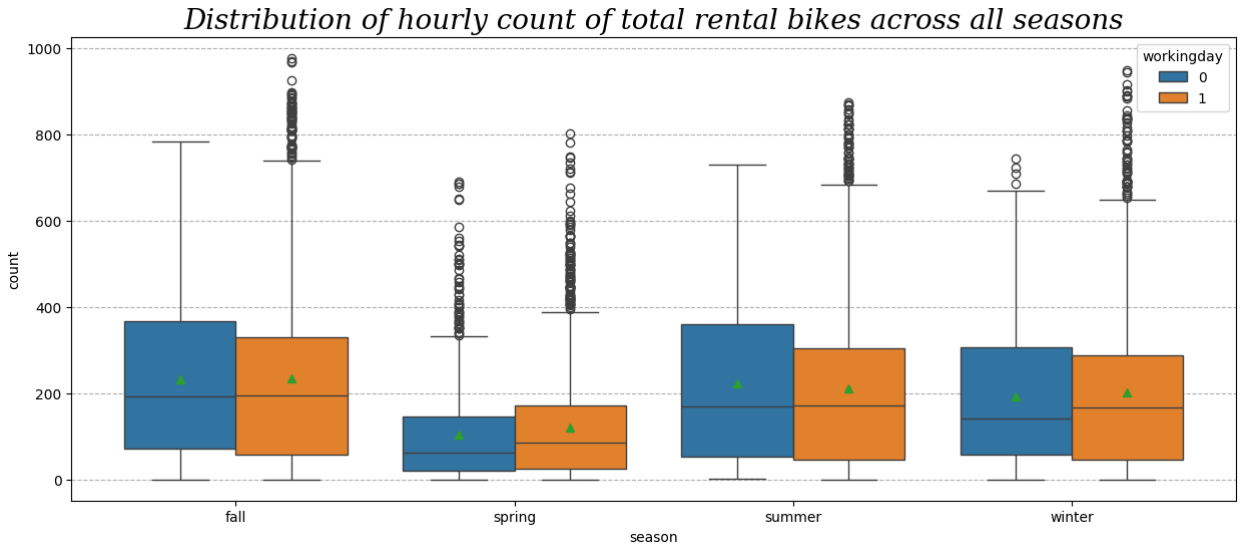


Observation

1. There is no outlier in the temp column.
2. There are few outliers present in humidity column.
3. There are many outliers present in each of the columns : windspeed, casual, registered, count.

Bivariate Analysis

```
plt.figure(figsize = (15, 6))
plt.title('Distribution of hourly count of total rental bikes across all
seasons',
          fontdict = {'size' : 20,
                      'style' : 'oblique',
                      'family' : 'serif'})
sns.boxplot(data = df, x = 'season', y = 'count', hue = 'workingday',
showmeans = True)
plt.grid(axis = 'y', linestyle = '--')
plt.plot()
```

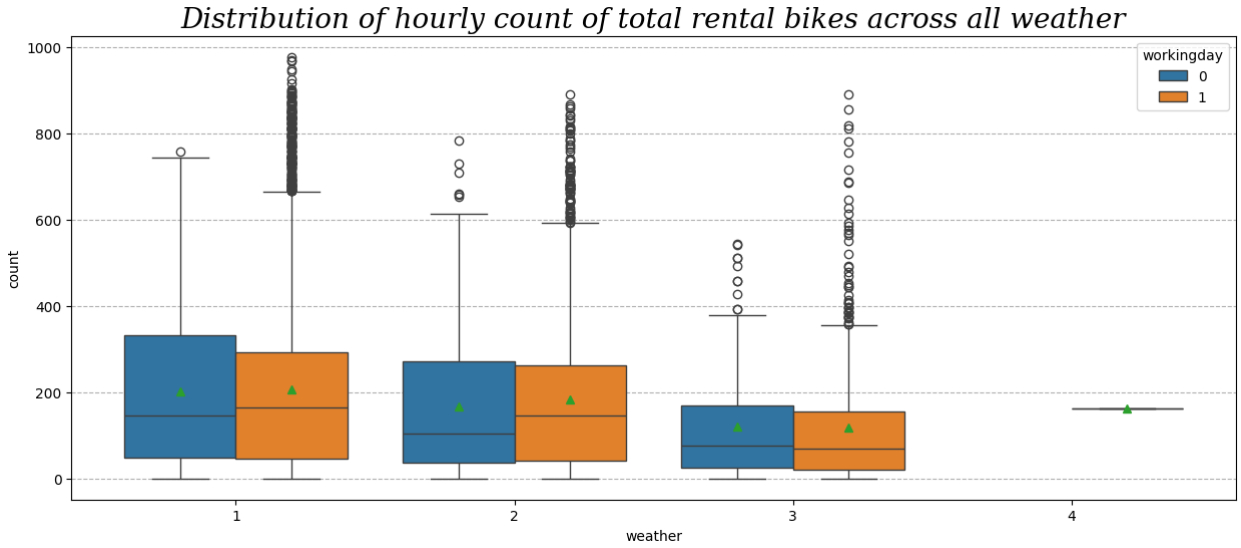


Observation

1. The hourly count of total rental bikes is higher in the fall season, followed by the summer and winter seasons. It is generally low in the spring season.

```
plt.figure(figsize = (15, 6))
plt.title('Distribution of hourly count of total rental bikes across all
weather',
          fontdict = {'size' : 20,
                      'style' : 'oblique',
                      'family' : 'serif'})
sns.boxplot(data = df, x = 'weather', y = 'count', hue = 'workingday',
showmeans = True)
```

```
plt.grid(axis = 'y', linestyle = '--')
plt.plot()
```



Observation

1. The hourly count of total rental bikes is higher in the clear and cloudy weather, followed by the misty weather and rainy weather. There are very few records for extreme weather conditions.

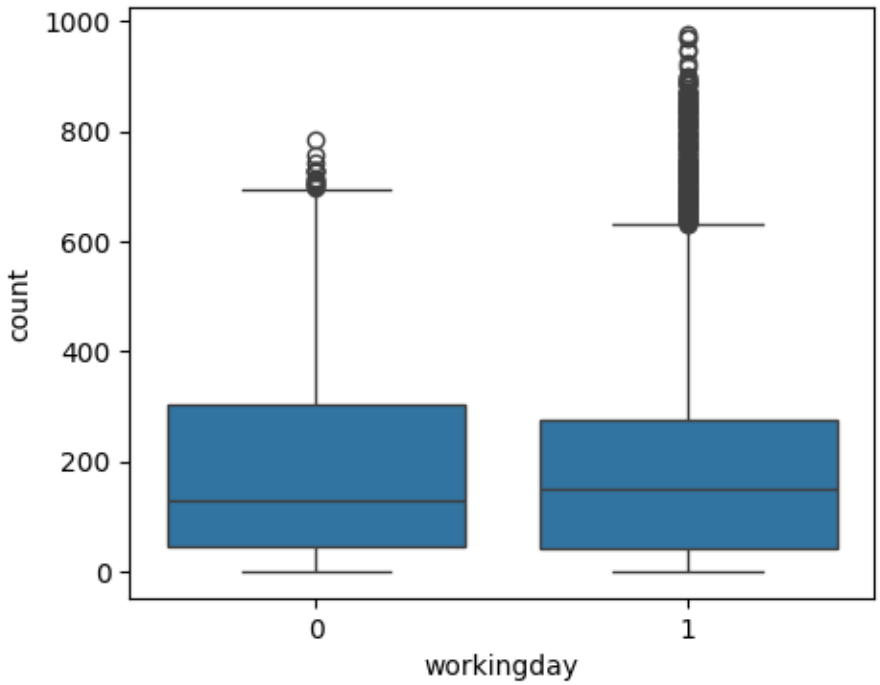
Now we will test on below points and secure the predictions.

1)Is there any effect of Working Day on the number of electric cycles rented?

```
# grouping workday and taking count of this and describing the data
df.groupby(by = 'workingday')['count'].describe()
```

workingday	count	mean	std	min	25%	50%	75%	max
0	3474.0	188.506621	173.724015	1.0	44.0	128.0	304.0	783.0
1	7412.0	193.011873	184.513659	1.0	41.0	151.0	277.0	977.0

```
plt.figure(figsize = (5, 4))
sns.boxplot(data = df, x = 'workingday', y = 'count')
plt.show()
```



STEP-1 : Set up Null Hypothesis

Null Hypothesis (H0) - Working Day does not have any effect on the number of electric cycles rented.

Alternate Hypothesis (HA) - Working Day has some effect on the number of electric cycles rented.

STEP-2 : Checking for basic assumptions for the hypothesis

1. Distribution check using **QQ Plot**
2. Homogeneity of Variances using **Levene's test**

STEP-3: Define Test statistics; Distribution of T under H0.

If the assumptions of T Test are met then we can proceed performing T Test for independent samples else we will perform the non-parametric test equivalent to T Test for independent sample i.e., Mann-Whitney U rank test for two independent samples.

STEP-4: Compute the p-value and fix value of alpha.

We set our **alpha to be 0.05.**

STEP-5: Compare p-value and alpha.

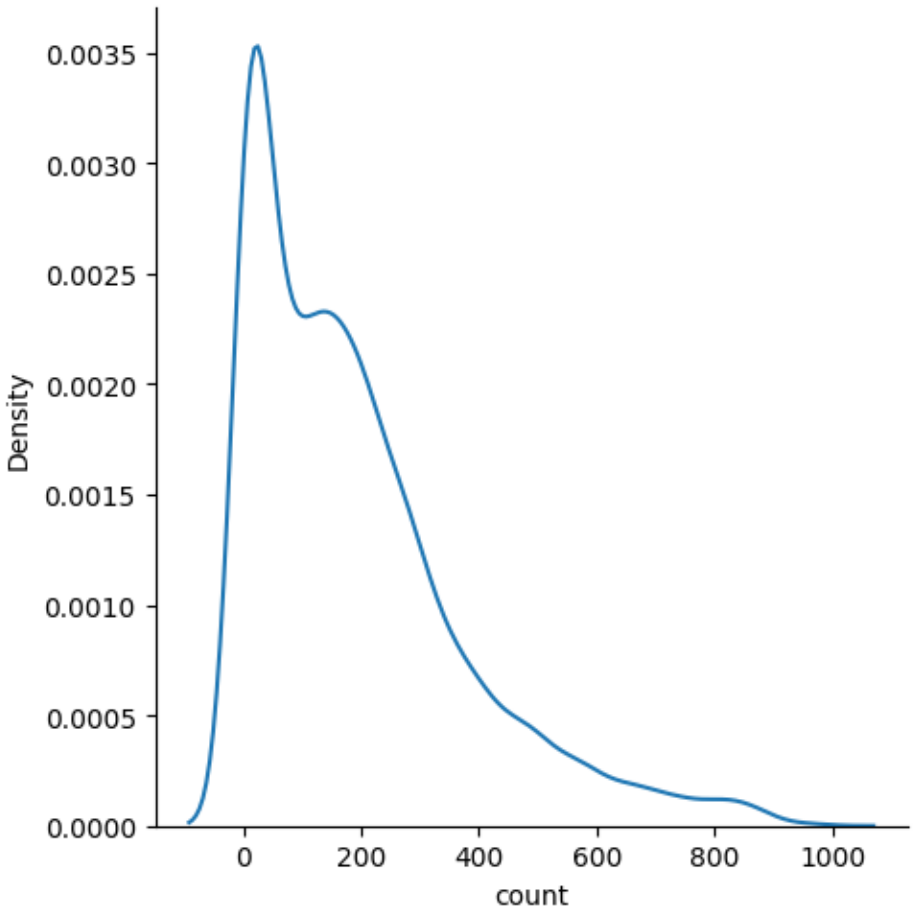
Based on p-value, we will accept or reject H0.

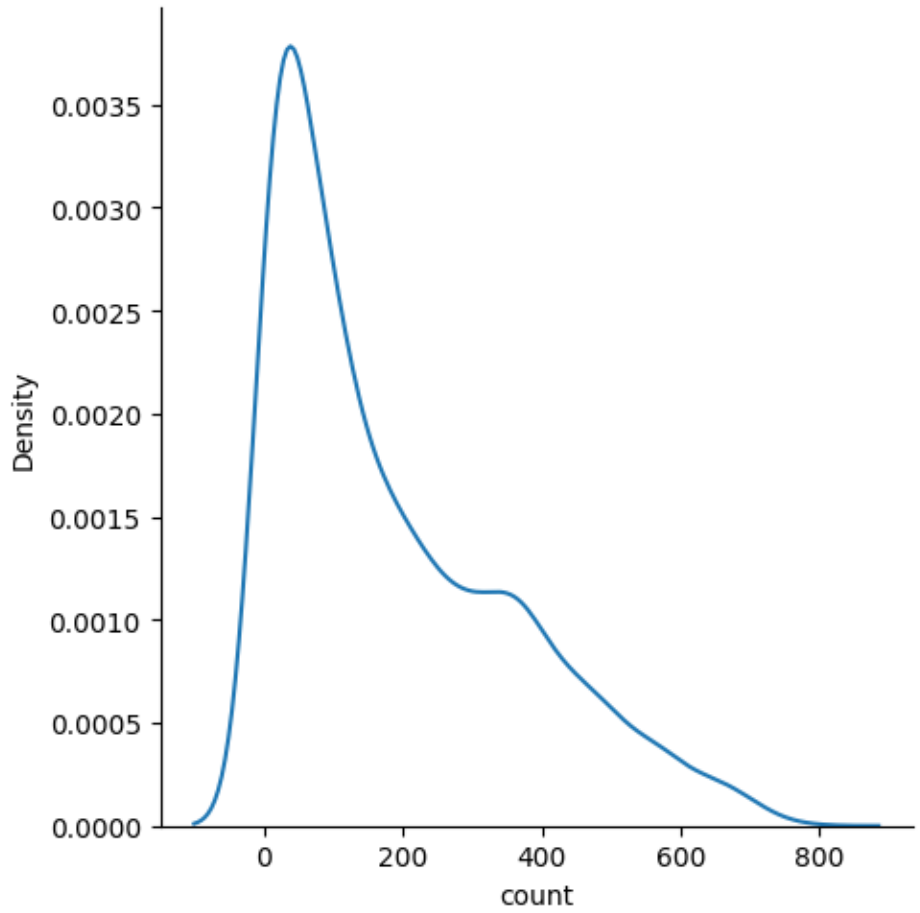
p-val > alpha : Accept H0

p-val < alpha : Reject H0

Visual Tests to know if the samples follow normal distribution

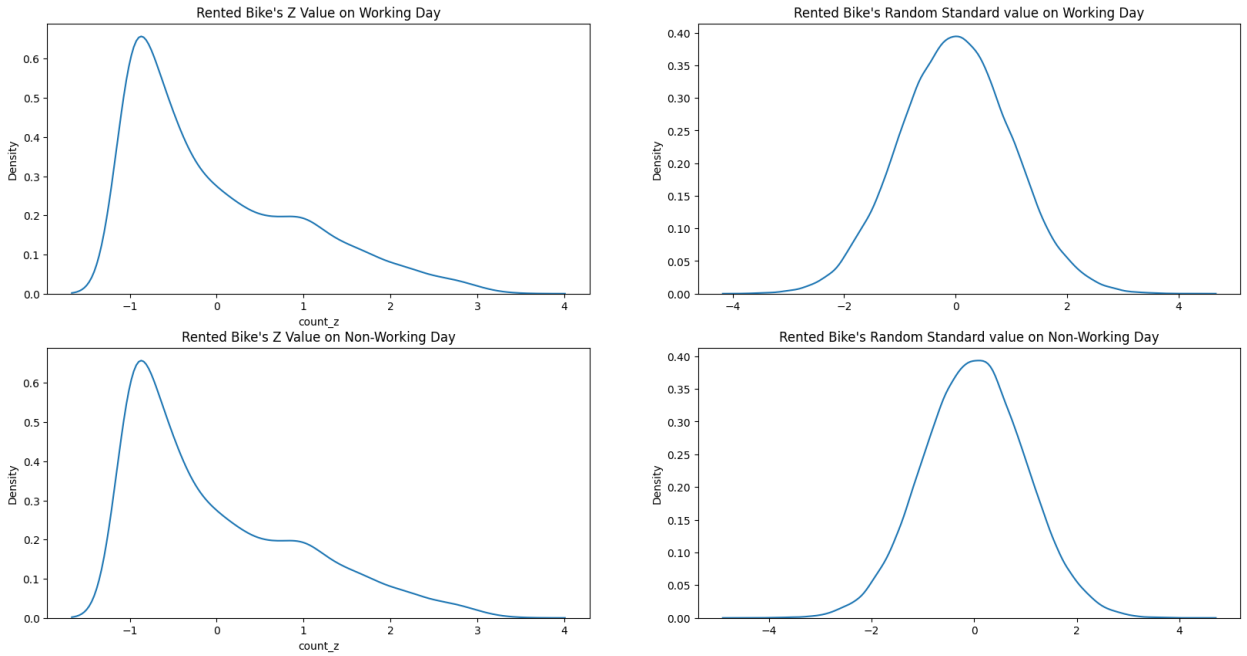
```
# first fetching the data on working days
workday = df[df['workingday'] == 1]

# fetching the data on non-working days
non_workday = df[df['workingday'] == 0]

plt.figure(figsize = (10,4))

sns.displot(data = workday, x = 'count', kind = 'kde')
sns.displot(data = non_workday, x = 'count', kind = 'kde')
plt.show()
```


0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
Density
0
200
400
600
800
1000
count




```
# will find out the Z value of count as per working day .
# will find it out standerd variation randomly for working day
workday['count_z'] = (workday['count'] -
workday['count'].mean())/workday['count'].std()
std_count_worday = np.random.normal(0,1,workday.shape)

# will find out the Z value of count as per non working day .
# will find it out standerd variation randomly for non working day
non_workday['count_z'] = (non_workday['count'] -
non_workday['count'].mean())/non_workday['count'].std()
std_count_non_workday = np.random.normal(0,1,non_workday.shape)

# setting plot figure size
plt.figure(figsize = (20, 10))

plt.subplot(2,2,1)
sns.distplot(workday['count_z'], hist = False)
plt.title("Rented Bike's Z Value on Working Day")
plt.subplot(2,2,2)
sns.distplot(std_count_worday, hist = False)
```

```
plt.title("Rented Bike's Random Standard value on Working Day")

plt.subplot(2,2,3)
sns.distplot(non_workday['count_z'], hist = False)
plt.title("Rented Bike's Z Value on Non-Working Day")
plt.subplot(2,2,4)
sns.distplot(std_count_non_workday, hist = False)
plt.title("Rented Bike's Random Standard value on Non-Working Day")
plt.show()
```



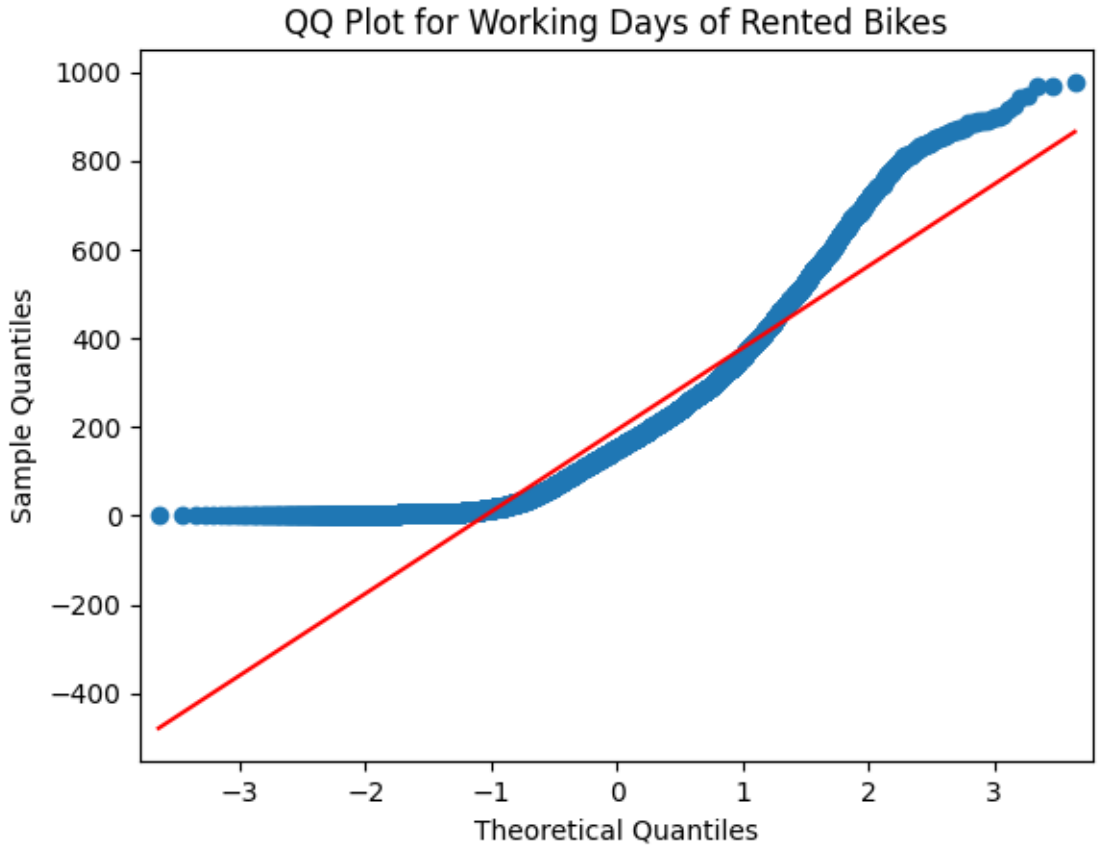
Observations

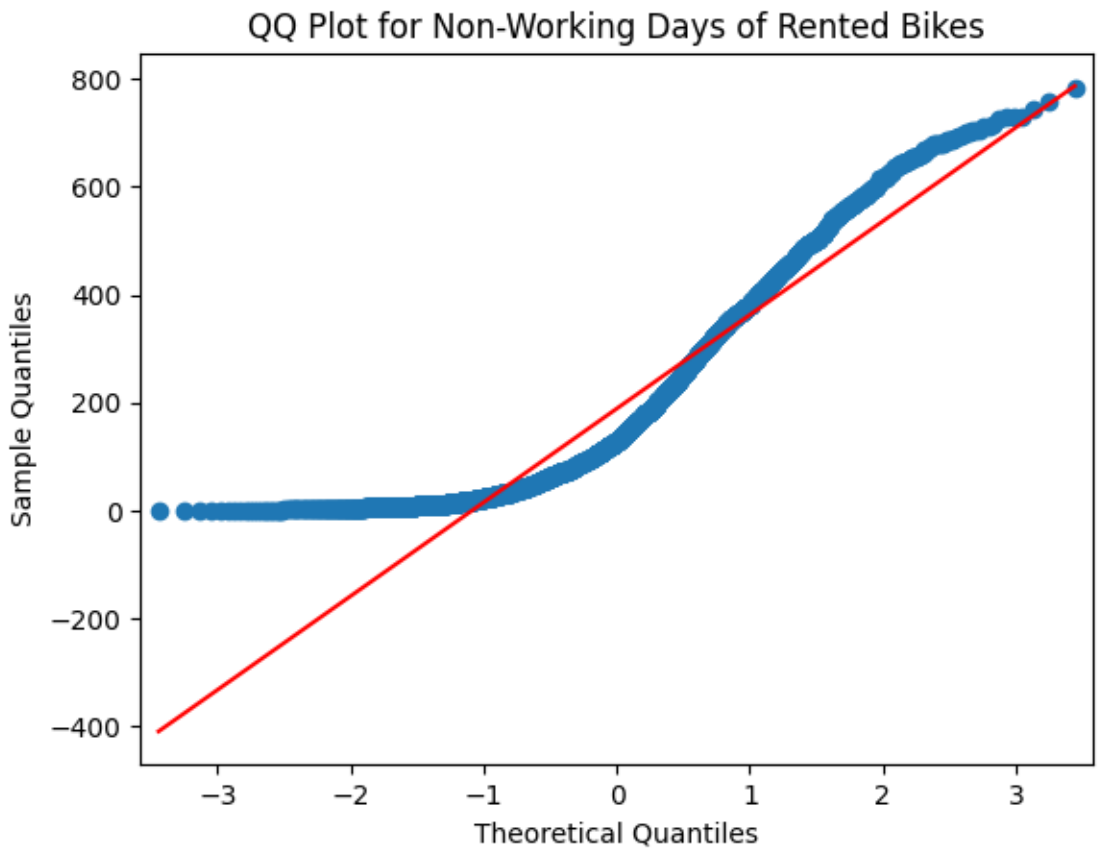
It can be inferred from the above plot that the distributions do not follow normal distribution.

Distribution check using QQ Plot

```
from statsmodels.graphics.gofplots import qqplot

plt.figure(figsize = (15,4))

# checking linearity of rented bikes for the working days
qqplot(workday['count'], line = 's')
plt.title("QQ Plot for Working Days of Rented Bikes")

# checking linearity of rented bikes for the non-working days
```

```
qqplot(non_workday['count'], line = 's')
plt.title("QQ Plot for Non-Working Days of Rented Bikes")

plt.show()
```





Observations

1. It can be inferred from the above plot that the distributions do not follow normal distribution.
2. It can be seen from the above plots that the samples do not come from normal distribution.

Applying Shapiro-Wilk test for normality

H0 : The sample follows normal distribution

Ha : The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

```
# now will check Shapiro Test
# for this test sample size must be <= 100
from scipy.stats import shapiro

# fetching workingday count
count_subset_workday = workday['count'].sample(100)
```

```
# fetching non workinday count
count_subset_non_workday = non_workday['count'].sample(100)

# finding shapiro test result for workday
test_stat_workday, pvalue_workday = shapiro(count_subset_workday)

# finding shapiro test result for non-workday
test_stat_non_workday, pvalue_non_workday =
shapiro(count_subset_non_workday)

print(f'pvalue is for working day: {pvalue_workday}')
# checking with pvalue if it is accespting or rejecting the null value for
working day
if pvalue_workday < 0.05 :
  print('The working day sample does not follow normal distribution.')
else :
  print('The working day sample follow normal distribution.')

print(f'pvalue is for non-working day: {pvalue_non_workday}')
# checking with pvalue if it is accespting or rejecting the null value for
non-working day
if pvalue_non_workday < 0.05 :
  print('The non-working day sample does not follow normal distribution.')
else :
  print('The non-working day sample follow normal distribution.')
```

pvalue is for working day: 3.7532683450081095e-07

The working day sample does not follow normal distribution.

pvalue is for non-working day: 1.2494801637785713e-07

The non-working day sample does not follow normal distribution.

Transforming the data using boxcox transformation and checking if the transformed data follows normal distribution.

```
from scipy.stats import boxcox
import warnings

# transforming data for working day
transform_wd = boxcox(workday['count'])[0]

# transforming data for non-working day
```

```
transform_nwd = boxcox(non_workday['count'])[0]

# Ignore the UserWarning
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    test_stat_workday, p_value_workday = shapiro(transform_wd)
    test_stat_non_workday, p_value_non_workday = shapiro(transform_nwd)

# checking pvalue with significant value and based on that will
accept/reject null
print(f'pvalue is for working day:{pvalue_workday}')
# checking with pvalue if it is accespting or rejecting the null value for
working day
if pvalue_workday < 0.05 :
  print('The working day sample does not follow normal distribution.')
else :
  print('The working day sample follow normal distribution.')

print(f'pvalue is for non-working day:{pvalue_non_workday}')
# checking with pvalue if it is accespting or rejecting the null value for
non-working day
if pvalue_non_workday < 0.05 :
  print('The non-working day sample does not follow normal distribution.')
else :
  print('The non-working day sample follow normal distribution.')
```

pvalue is for working day:3.7532683450081095e-07

The working day sample does not follow normal distribution.

pvalue is for non-working day:1.2494801637785713e-07

The non-working day sample does not follow normal distribution.

Observations

Even after applying the boxcox transformation on each of the "workingday" and "non_workingday" data, the samples do not follow normal distribution.

Homogeneity of Variances using Lavene's test.

```
# Null Hypothesis(H0) - Homogenous Variance
# Alternate Hypothesis(HA) - Non Homogenous Variance

from scipy.stats import levene
```

```
test_stas, p_value = levene(workday['count'],non_workday['count'])

print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance.')
else:
    print('The samples have Homogenous Variance. ')
```

p-value 0.9437823280916695

Observations

The samples have Homogenous Variance.

Observations

Since the samples are not normally distributed, T-Test cannot be applied here, we can perform its non-parametric equivalent test i.e., Mann-Whitney U rank test for two independent samples.

```
# Ho : Mean no.of electric cycles rented is same for working and non-working days
# Ha : Mean no.of electric cycles rented is not same for working and non-working days
# Assuming significance Level to be 0.05
# Test statistics : Mann-Whitney U rank test for two independent samples

from scipy.stats import mannwhitneyu

test_stas, p_value = mannwhitneyu(workday['count'],non_workday['count'])


print('P-value :',p_value)
if p_value < 0.05:
    print('Mean no.of electric cycles rented is not same for working and non-working days.')
else:
    print('Mean no.of electric cycles rented is same for working and non-working days.')
```

P-value : 0.9679139953914079

Mean no.of electric cycles rented is same for working and non-working days.

Conclusion of the Test:

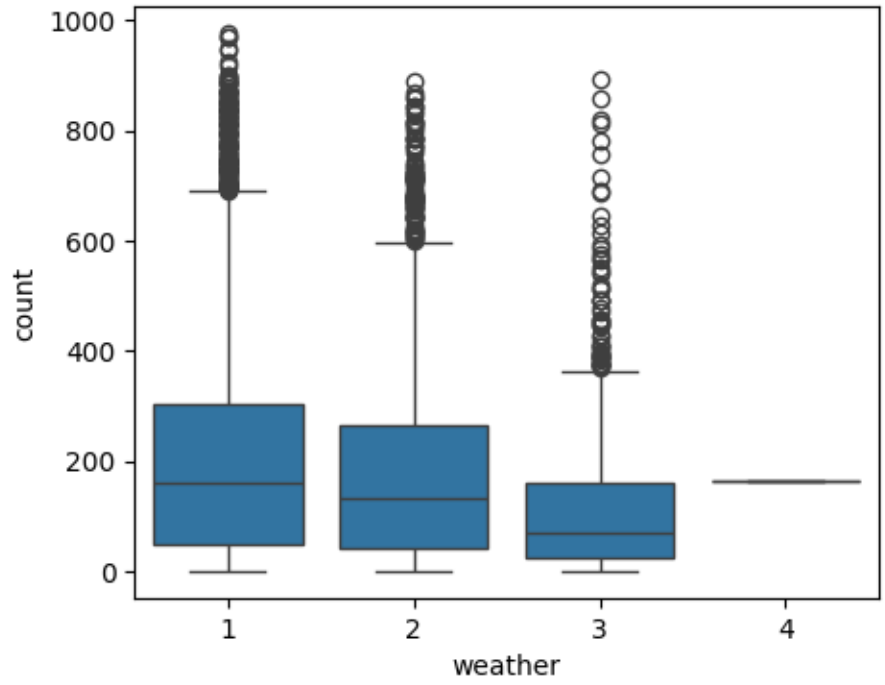
Therefore, the mean hourly count of the total rental bikes is statistically same for both working and non- working days.

2) Is the number of cycles rented is similar or different in different weather ?

```
# grouping weather and taking count of this and describing the data
df.groupby(by = 'weather')['count'].describe()
```

weather	count	mean	std	min	25%	50%	75%	max
1	7192.0	205.236791	187.959566	1.0	48.0	161.0	305.0	977.0
2	2834.0	178.955540	168.366413	1.0	41.0	134.0	264.0	890.0
3	859.0	118.846333	138.581297	1.0	23.0	71.0	161.0	891.0
4	1.0	164.000000	NaN	164.0	164.0	164.0	164.0	164.0

```
plt.figure(figsize = (5, 4))
sns.boxplot(data = df, x = 'weather', y = 'count')
plt.show()
```



STEP-1 : Set up Null Hypothesis

Null Hypothesis (H0) - Mean of cycle rented per hour is same for weather 1, 2 and 3.

Alternate Hypothesis (HA) - Mean of cycle rented per hour is not same for weather 1, 2 and 3.

Note : We will not check weather 4 as it has only one data. We wont be \able to any kind of test.

STEP-2 : Checking for basic assumptions for the hypothesis

3. Distribution check using **QQ Plot**
4. Homogeneity of Variances using **Levene's test**

STEP-3: Define Test statistics; Distribution of T under H0.

The test statistic for a One-Way ANOVA is denoted as F. For an independent variable with k groups, the F statistic evaluates whether the group means are significantly different.

STEP-4: Decide the kind of test.

We will be performing **right tailed f-test**

STEP-5: Compute the p-value and fix value of alpha.

We set our **alpha to be 0.05.**

STEP-6: Compare p-value and alpha.

Based on p-value, we will accept or reject H0.

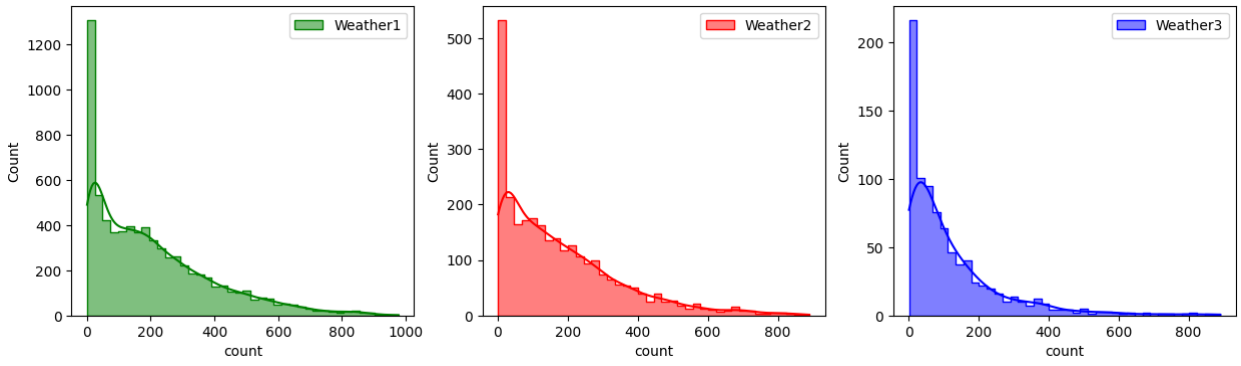
p-val > alpha : Accept H0

p-val < alpha : Reject H0

Visual Tests to know if the samples follow normal distribution.

```
# first fetching the data on weather based on 1
weather1 = df[df['weather'] == 1]

# first fetching the data on weather based on 2
weather2 = df[df['weather'] == 2]

# first fetching the data on weather based on 3
weather3 = df[df['weather'] == 3]
```

```
# first fetching the data on weather based on 4
weather4 = df[df['weather'] == 4]

plt.figure(figsize = (15,4))

plt.subplot(1, 3, 1)
sns.histplot(weather1['count'], bins = 40,
             element = 'step', color = 'green', kde = True, label =
'Weather1')
plt.legend()

plt.subplot(1, 3, 2)
sns.histplot(weather2['count'], bins = 40,
              element = 'step', color = 'red', kde = True, label =
'Weather2')
plt.legend()

plt.subplot(1, 3, 3)
sns.histplot(weather3['count'], bins = 40,
              element = 'step', color = 'blue', kde = True, label =
'Weather3')
plt.legend()
plt.show()
```



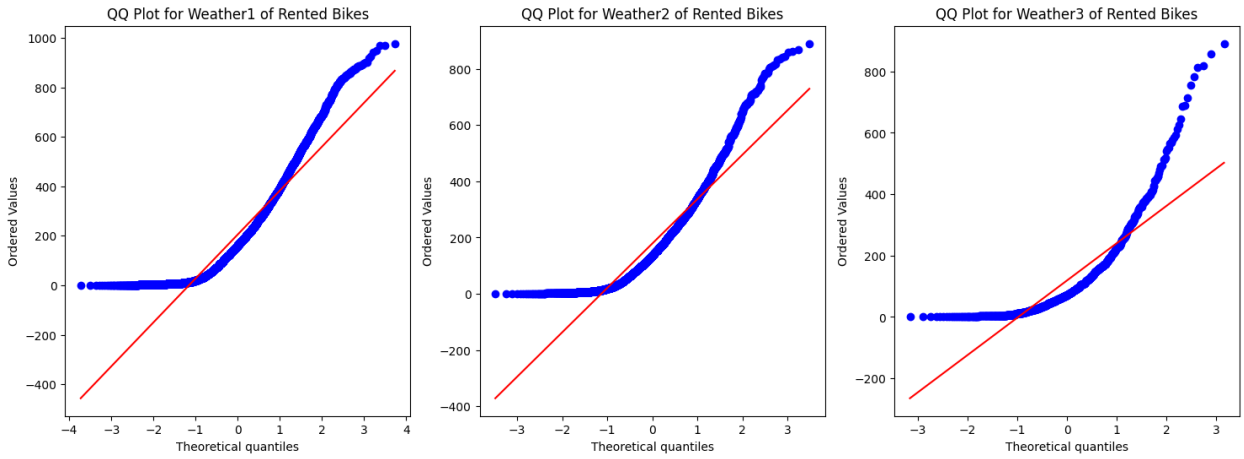
Observations

It can be inferred from the above plot that the distributions do not follow normal distribution.

Distribution check using QQ Plot

```
from statsmodels.graphics.gofplots import qqplot
```

```
from scipy.stats import probplot

plt.figure(figsize = (18,6))

# checking linearity of rented bikes for the weather1
plt.subplot(1,3,1)
probplot(weather1['count'], plot = plt, dist = 'norm')
plt.title("QQ Plot for Weather1 of Rented Bikes")

# checking linearity of rented bikes for the weather2
plt.subplot(1,3,2)
probplot(weather2['count'], plot = plt, dist = 'norm')
plt.title("QQ Plot for Weather2 of Rented Bikes")

# checking linearity of rented bikes for the weather3
plt.subplot(1,3,3)
probplot(weather3['count'], plot = plt, dist = 'norm')
plt.title("QQ Plot for Weather3 of Rented Bikes")

plt.show()
```



Observations

3. It can be inferred from the above plot that the distributions do not follow normal distribution.
4. It can be seen from the above plots that the samples do not come from normal distribution.

Applying Shapiro-Wilk test for normality

H0 : The sample follows normal distribution

Ha : The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

```
# now will check Shapiro Test
# for this test sample size must be <= 100
from scipy.stats import shapiro

# fetching weather1 count
count_subset_weather1 = weather1['count'].sample(100)

# fetching weather2 count
count_subset_weather2 = weather2['count'].sample(100)

# fetching weather3 count
count_subset_weather3 = weather3['count'].sample(100)

# finding shapiro test result for weather1
test_stat_weather1, pvalue_weather1 = shapiro(count_subset_weather1)

# finding shapiro test result for non-workday
test_stat_weather2, pvalue_weather2 = shapiro(count_subset_weather2)

# finding shapiro test result for weather3
test_stat_weather3, pvalue_weather3 = shapiro(count_subset_weather3)

print(f'pvalue is for weather1:{pvalue_weather1}')
# checking with pvalue if it is accespting or rejecting the null value for
weather1 day
if pvalue_weather1 < 0.05 :
  print('The Weather1 sample does not follow normal distribution')
else :
  print('The Weather1 sample follow normal distribution')

print(f'pvalue is for weather1:{pvalue_weather2}')
# checking with pvalue if it is accespting or rejecting the null value for
weather2 day
if pvalue_weather2 < 0.05 :
  print('The Weather2 sample does not follow normal distribution')
else :
  print('The Weather2 sample follow normal distribution')

print(f'pvalue is for weather1:{pvalue_weather3}')
# checking with pvalue if it is accespting or rejecting the null value for
weather3 day
if pvalue_weather3 < 0.05 :
```

```
  print('The Weather3 sample does not follow normal distribution')
else :
  print('The Weather3 sample follow normal distribution')
```

pvalue is for weather1:6.666944329936086e-08

The Weather1 sample does not follow normal distribution

pvalue is for weather1:5.2838673582300544e-05

The Weather2 sample does not follow normal distribution

pvalue is for weather1:3.508368873195167e-10

The Weather3 sample does not follow normal distribution

Transforming the data using boxcox transformation and checking if the transformed data follows normal distribution.

```
from scipy.stats import boxcox
import warnings

# transforming data for weather1
transform_wd1 = boxcox(weather1['count'])[0]

# transforming data for weather2
transform_wd2 = boxcox(weather2['count'])[0]

# transforming data for weather3
transform_wd3 = boxcox(weather3['count'])[0]

# Ignore the UserWarning
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    test_stat_weather1, p_value_weather1 = shapiro(transform_wd1)
    test_stat_weather2, p_value_weather2 = shapiro(transform_wd2)
    test_stat_weather3, p_value_weather3 = shapiro(transform_wd3)

# checking pvalue with significant value and based on that will
accept/reject null
print(f'pvalue is for weather1:{p_value_weather1}')
# checking with pvalue if it is accespting or rejecting the null value for
working day
if pvalue_weather1 < 0.05 :
```

```
  print('The weather1 sample does not follow normal distribution.')
else :
  print('The weather1 sample follow normal distribution.')

# checking pvalue with significant value and based on that will
accept/reject null
print(f'pvalue is for weather2:{p_value_weather2}')
# checking with pvalue if it is accespting or rejecting the null value for
working day
if pvalue_weather2 < 0.05 :
  print('The weather2 sample does not follow normal distribution.')
else :
  print('The weather2 sample follow normal distribution.')

# checking pvalue with significant value and based on that will
accept/reject null
print(f'pvalue is for weather3:{p_value_weather3}')
# checking with pvalue if it is accespting or rejecting the null value for
working day
if pvalue_weather3 < 0.05 :
  print('The weather3 sample does not follow normal distribution.')
else :
  print('The weather3 sample follow normal distribution.')
```

pvalue is for weather1:2.061217589223373e-32

The weather1 sample does not follow normal distribution.

pvalue is for weather2:1.9216098393369846e-19

The weather2 sample does not follow normal distribution.

pvalue is for weather3:1.4133181593933841e-06

The weather3 sample does not follow normal distribution.

Observations

Even after applying the boxcox transformation on each of the "weather1" , "weather2" and "weather3" data, the samples do not follow normal distribution.

Homogeneity of Variances using Lavene's test.

```
# Null Hypothesis(H0) - Homogenous Variance
# Alternate Hypothesis(HA) - Non Homogenous Variance

from scipy.stats import levene
```

```
test_stas, p_value =
levene(weather1['count'],weather2['count'],weather3['count'])

print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

p-value 6.198278710731511e-36

The samples do not have Homogenous Variance

Observations

Since the samples are not normally distributed and do not have the same variance, f_oneway test cannot be performed here, we can perform its non parametric equivalent test i.e., Kruskal-Wallis H-test for independent samples.

```
# Ho : Mean no.of electric cycles rented is same for all the weather
conditions
# Ha : Mean no.of electric cycles rented is not same for all the weather
conditions
# Assuming significance Level to be 0.05
# Test statistics : Kruskal-Wallis H-test for two independent samples

from scipy.stats import kruskal
test_stas, p_value =
kruskal(weather1['count'],weather2['count'],weather3['count'])

print('P-value :',p_value)
if p_value < 0.05:
    print('Mean no.of electric cycles rented is not same for all the
weather conditions')
else:
    print('Mean no.of electric cycles rented is same for all the weather
conditions')
```

P-value : 3.122066178659941e-45

Mean no.of electric cycles rented is not same for all the weather conditions

Conclusion of the Test:

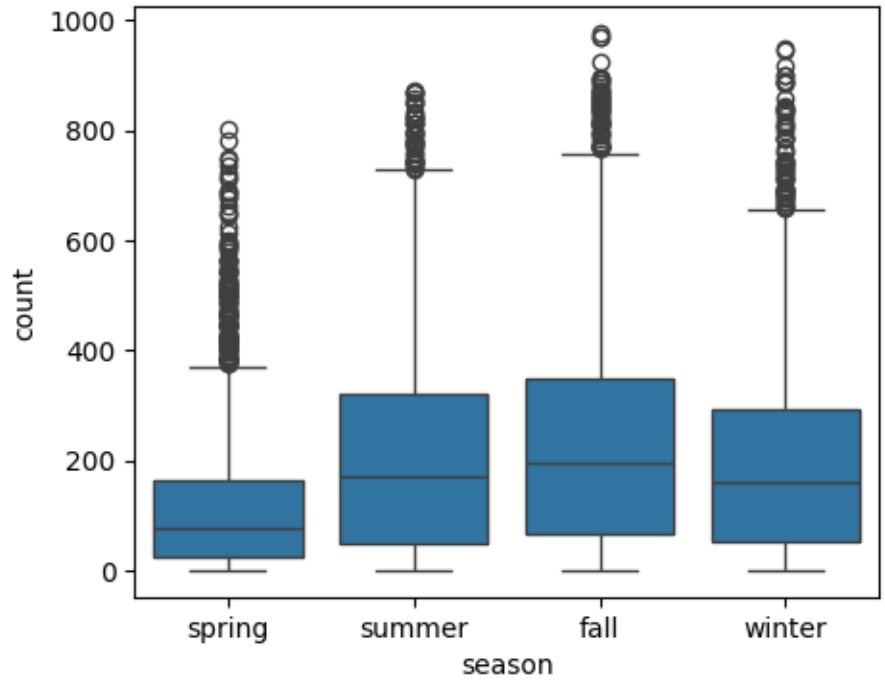
Therefore, the mean hourly count of the total rental bikes is statistically different for different weather conditions.

2) Is the number of cycles rented is similar or different in different seasons ?

```
# grouping season and taking count of this and describing the data
df.groupby(by = 'season')['count'].describe()
```

season	count	mean	std	min	25%	50%	75%	max
spring	2686.0	116.343261	125.273974	1.0	24.0	78.0	164.0	801.0
summer	2733.0	215.251372	192.007843	1.0	49.0	172.0	321.0	873.0
fall	2733.0	234.417124	197.151001	1.0	68.0	195.0	347.0	977.0
winter	2734.0	198.988296	177.622409	1.0	51.0	161.0	294.0	948.0

```
plt.figure(figsize = (5, 4))
sns.boxplot(data = df, x = 'season', y = 'count')
plt.show()
```



STEP-1 : Set up Null Hypothesis

Null Hypothesis (H0) - Mean of cycle rented per hour is same for different seasons.

Alternate Hypothesis (HA) - Mean of cycle rented per hour is different for different seasons.

STEP-2 : Checking for basic assumptions for the hypothesis

5. Distribution check using **QQ Plot**
6. Homogeneity of Variances using **Levene's test**

STEP-3: Define Test statistics; Distribution of T under H0.

The test statistic for a One-Way ANOVA is denoted as F. For an independent variable with k groups, the F statistic evaluates whether the group means are significantly different.

STEP-4: Decide the kind of test.

We will be performing **right tailed f-test**

STEP-5: Compute the p-value and fix value of alpha.

We set our **alpha to be 0.05.**

STEP-6: Compare p-value and alpha.

Based on p-value, we will accept or reject H0.

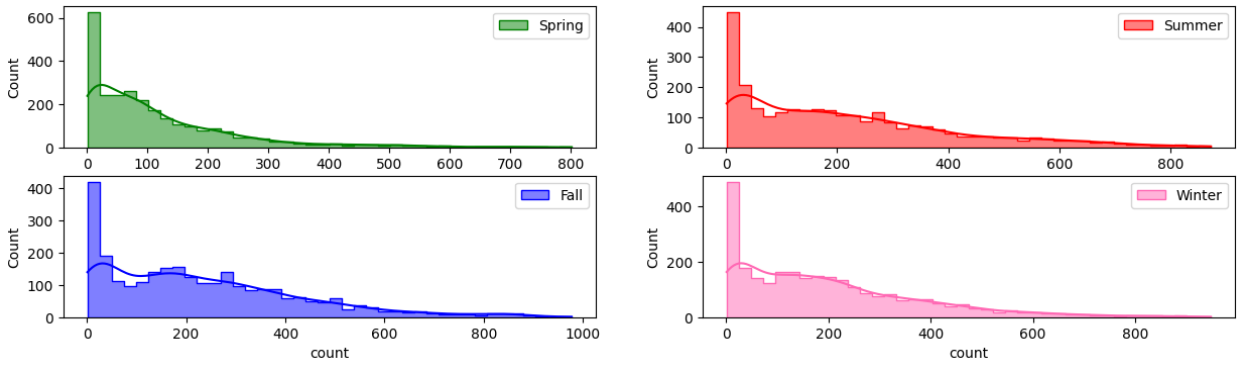
p-val > alpha : Accept H0

p-val < alpha : Reject H0

Visual Tests to know if the samples follow normal distribution.

```
# 1: spring, 2: summer, 3: fall, 4: winter
# first fetching the data on weather based on 1
df_spring = df[df['season'] == 'spring']

# first fetching the data on weather based on 2
df_summer = df[df['season'] == 'summer']

# first fetching the data on weather based on 3
df_fall = df[df['season'] == 'fall']

# first fetching the data on weather based on 4
df_winter = df[df['season'] == 'winter']

plt.figure(figsize = (15,4))
```

```
plt.subplot(2, 2, 1)
sns.histplot(df_spring['count'], bins = 40,
             element = 'step', color = 'green', kde = True, label =
'Spring')
plt.legend()

plt.subplot(2, 2, 2)
sns.histplot(df_summer['count'], bins = 40,
             element = 'step', color = 'red', kde = True, label =
'Summer')
plt.legend()

plt.subplot(2, 2, 3)
sns.histplot(df_fall['count'], bins = 40,
             element = 'step', color = 'blue', kde = True, label = 'Fall')
plt.legend()

plt.subplot(2, 2, 4)
sns.histplot(df_winter['count'], bins = 40,
             element = 'step', color = 'hotpink', kde = True, label =
'Winter')
plt.legend()
plt.show()
```



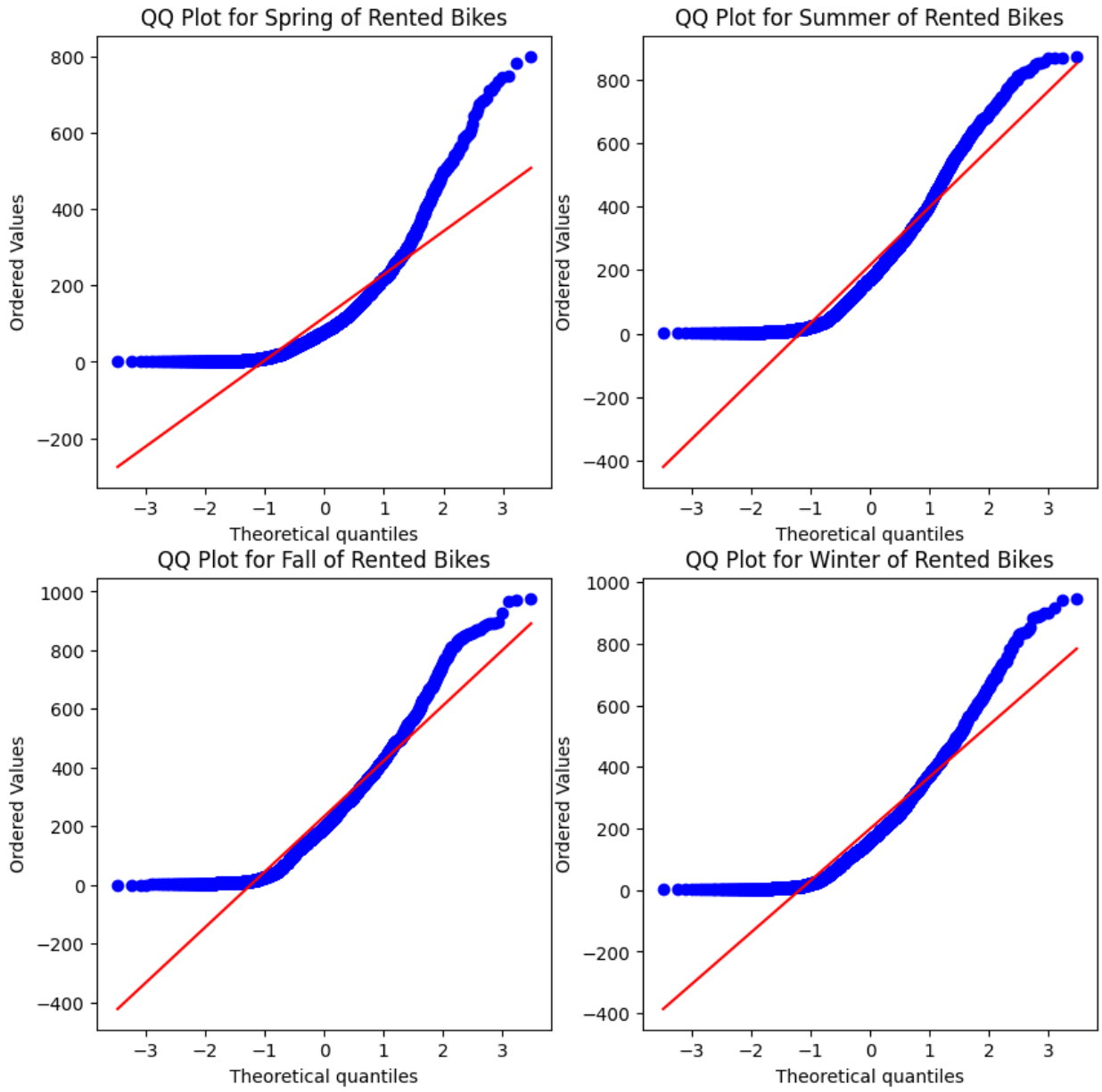
Observations

It can be inferred from the above plot that the distributions do not follow normal distribution.

Distribution check using QQ Plot

```
from statsmodels.graphics.gofplots import qqplot
```

```
from scipy.stats import probplot

plt.figure(figsize = (10,10))

# checking linearity of rented bikes for the spring
plt.subplot(2,2,1)
probplot(df_spring['count'], plot = plt, dist = 'norm')
plt.title("QQ Plot for Spring of Rented Bikes")

# checking linearity of rented bikes for the summer
plt.subplot(2,2,2)
probplot(df_summer['count'], plot = plt, dist = 'norm')
plt.title("QQ Plot for Summer of Rented Bikes")

# checking linearity of rented bikes for the fall
plt.subplot(2,2,3)
probplot(df_fall['count'], plot = plt, dist = 'norm')
plt.title("QQ Plot for Fall of Rented Bikes")

# checking linearity of rented bikes for the fall
plt.subplot(2,2,4)
probplot(df_winter['count'], plot = plt, dist = 'norm')
plt.title("QQ Plot for Winter of Rented Bikes")

plt.show()
```



Observations

5. It can be inferred from the above plot that the distributions do not follow normal distribution.
6. It can be seen from the above plots that the samples do not come from normal distribution.

Applying Shapiro-Wilk test for normality

H0 : The sample follows normal distribution

Ha : The sample does not follow normal distribution

alpha = 0.05

Test Statistics : Shapiro-Wilk test for normality

```
# now will check Shapiro Test
# for this test sample size must be <= 100
from scipy.stats import shapiro

# fetching spring count
count_subset_spring = df_spring['count'].sample(100)

# fetching summer count
count_subset_summer = df_summer['count'].sample(100)

# fetching fall count
count_subset_fall = df_fall['count'].sample(100)

# fetching winter count
count_subset_winter = df_winter['count'].sample(100)

# finding shapiro test result for spring
test_stat_spring, pvalue_spring = shapiro(count_subset_spring)

# finding shapiro test result for non-workday
test_stat_summer, pvalue_summer = shapiro(count_subset_summer)

# finding shapiro test result for fall
test_stat_fall, pvalue_fall = shapiro(count_subset_fall)

# finding shapiro test result for fall
test_stat_winter, pvalue_winter = shapiro(count_subset_winter)

print(f'pvalue is for spring:{pvalue_spring}')
# checking with pvalue if it is accespting or rejecting the null value for
spring day
if pvalue_spring < 0.05 :
  print('The Spring sample does not follow normal distribution')
else :
  print('The Spring sample follow normal distribution')

print(f'pvalue is for Summer:{pvalue_summer}')
# checking with pvalue if it is accespting or rejecting the null value for
summer day
if pvalue_summer < 0.05 :
  print('The Summer sample does not follow normal distribution')
else :
```

```
  print('The Summer sample follow normal distribution')

print(f'pvalue is for Fall:{pvalue_fall}')
# checking with pvalue if it is accespting or rejecting the null value for
Fall day
if pvalue_fall < 0.05 :
  print('The Fall sample does not follow normal distribution')
else :
  print('The Fall sample follow normal distribution')

print(f'pvalue is for Winter:{pvalue_winter}')
# checking with pvalue if it is accespting or rejecting the null value for
Winter day
if pvalue_winter < 0.05 :
  print('The Winter sample does not follow normal distribution')
else :
  print('The Winter sample follow normal distribution')
```

pvalue is for spring:1.9336855727747349e-10

The Spring sample does not follow normal distribution

pvalue is for Summer:5.904502540943213e-05

The Summer sample does not follow normal distribution

pvalue is for Fall:2.887006012031179e-08

The Fall sample does not follow normal distribution

pvalue is for Winter:3.7589913404190156e-08

The Winter sample does not follow normal distribution

Transforming the data using boxcox transformation and checking if the transformed data follows normal distribution.

```
from scipy.stats import boxcox
import warnings

# transforming data for spring
transform_spring = boxcox(df_spring['count'])[0]

# transforming data for summer
transform_summer = boxcox(df_summer['count'])[0]

# transforming data for fall
```

```
transform_fall = boxcox(df_fall['count'])[0]

# transforming data for winter
transform_winter = boxcox(df_winter['count'])[0]

# Ignore the UserWarning
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    test_stat_spring, p_value_spring = shapiro(transform_spring)
    test_stat_summer, p_value_summer = shapiro(transform_summer)
    test_stat_fall, p_value_fall = shapiro(transform_fall)
    test_stat_winter, p_value_winter = shapiro(transform_winter)

# checking pvalue with significant value and based on that will
accept/reject null
print(f'pvalue is for spring:{p_value_spring}')
# checking with pvalue if it is accespting or rejecting the null value for
working day
if pvalue_spring < 0.05 :
  print('The spring sample does not follow normal distribution.')
else :
  print('The spring sample follow normal distribution.')

# checking pvalue with significant value and based on that will
accept/reject null
print(f'pvalue is for summer:{p_value_summer}')
# checking with pvalue if it is accespting or rejecting the null value for
working day
if pvalue_summer < 0.05 :
  print('The summer sample does not follow normal distribution.')
else :
  print('The summer sample follow normal distribution.')

# checking pvalue with significant value and based on that will
accept/reject null
print(f'pvalue is for fall:{p_value_fall}')
# checking with pvalue if it is accespting or rejecting the null value for
working day
if pvalue_fall < 0.05 :
  print('The fall sample does not follow normal distribution.')
else :
  print('The fall sample follow normal distribution.')

# checking pvalue with significant value and based on that will
accept/reject null
```

```
print(f'pvalue is for winter:{p_value_winter}')
# checking with pvalue if it is accespting or rejecting the null value for
working day
if pvalue_winter < 0.05 :
  print('The winter sample does not follow normal distribution.')
else :
  print('The winter sample follow normal distribution.')
```

pvalue is for spring:1.7082116755999925e-17

The spring sample does not follow normal distribution.

pvalue is for summer:2.7910560207702335e-22

The summer sample does not follow normal distribution.

pvalue is for fall:3.6319999210910884e-22

The fall sample does not follow normal distribution.

pvalue is for winter:6.342709865441161e-21

The winter sample does not follow normal distribution.

Observations

Even after applying the boxcox transformation on each of the "spring" , "summer", “fall” and "winter" data, the samples do not follow normal distribution.

Homogeneity of Variances using Lavene's test.

```
# Null Hypothesis(H0) - Homogenous Variance
# Alternate Hypothesis(HA) - Non Homogenous Variance

from scipy.stats import levene
test_stas, p_value =
levene(df_spring['count'],df_summer['count'],df_fall['count'],df_winter['c
ount'])

print('p-value', p_value)
if p_value < 0.05:
    print('The samples do not have  Homogenous Variance')
else:
    print('The samples have Homogenous Variance ')
```

p-value 1.0147116860043298e-118

The samples do not have Homogenous Variance

Observations

Since the samples are not normally distributed and do not have the same variance, f_oneway test cannot be performed here, we can perform its non parametric equivalent test i.e., Kruskal-Wallis H-test for independent samples.

```
# Ho : Mean no.of electric cycles rented is same for all the season
# Ha : Mean no.of electric cycles rented is not same for all the season
# Assuming significance Level to be 0.05
# Test statistics : Kruskal test for two independent samples

from scipy.stats import kruskal
test_stas, p_value =
kruskal(df_spring['count'],df_summer['count'],df_fall['count'],df_winter['
count'])

print('P-value :',p_value)
if p_value < 0.05:
    print('Mean no.of electric cycles rented is different for all the
Seasons. ')
else:
    print('Mean no.of electric cycles rented is same for all the
seasons.')
```

P-value : 2.479008372608633e-151

Mean no.of electric cycles rented is different for all the Seasons.

Conclusion of the Test:

Therefore, the mean hourly count of the total rental bikes is statistically different for different seasons.

4) Is weather dependent on the season?

```
df[['weather', 'season']].describe()
```

	weather	season
count	10886	10886
unique	4	4
top	1	winter
freq	7192	2734

It is clear from the above statistical description that both 'weather' and 'season' features are categorical in nature.

STEP-1 : Set up Null Hypothesis

Null Hypothesis (H0) - weather is not dependent/independent of season.

Alternate Hypothesis (HA) - weather is dependent of seasons.

STEP-2: Define Test statistics

Since we have two categorical features, **the Chi- square test** is applicable here. Under H0, the test statistic should follow **Chi-Square Distribution**.

STEP-3: Checking for basic assumptons for the hypothesis (Non-Parametric Test)

The data in the cells should be **frequencies**, or **counts** of cases.

The levels (or categories) of the variables are **mutually exclusive**. That is, a particular subject fits into one and only one level of each of the variables.

There are 2 variables, and both are **measured** as **categories**.

The value of the cell expected should be 5 or more in at least 80% of the cells, and no cell should have an expected of less than one (3).

STEP-4: Compute the p-value and fix value of alpha.

we will be computing the **chi square-test p-value** using the chi2_**contingency** function using **scipy.stats**. We set our **alpha** to be **0.05.**

STEP-5: Compare p-value and alpha.

Based on p-value, we will **accept or reject H0**.

p-val > alpha : Accept H0

p-val < alpha : Reject H0

The **Chi-square** statistic is a **non-parametric** (distribution free) tool designed to analyze group differences when the dependent variable is measured at a nominal level. Like all non-parametric statistics, the Chi-square is robust with respect to the distribution of the data. Specifically, it does not require equality of variances among the study groups or homoscedasticity in the data.

```
crosstab = pd.crosstab(columns = df['weather'],
            index = df['season'],
            values= df['count'],
            aggfunc=np.sum)
crosstab
```

weather	1	2	3	4
season				
spring	223009	76406	12919	164
summer	426350	134177	27755	0
fall	470116	139386	31160	0
winter	356588	157191	30255	0

Since the above contingency table has one column in which the count of the rented electric vehicle is less than 5 in most of the cells, we can remove the weather 4 and then proceed further.

```
crosstab = pd.crosstab(columns = df[df['weather'] != 4 ]['weather'],
            index = df['season'],
            values= df['count'],
            aggfunc=np.sum).loc[:,:3]
crosstab
```

weather	1	2	3
season			
spring	223009	76406	12919
summer	426350	134177	27755
fall	470116	139386	31160
winter	356588	157191	30255

```
from scipy.stats import chi2_contingency
chitest_statics_value, p_value, DOF, Expected_freq =
chi2_contingency(crosstab)
print(f"Statistics value from the test: {chitest_statics_value}")
print(f'P_Value of the test is: {p_value}')
print(f'Degree of freedom from the test: {DOF}')
print('*' * 50)
print(f'Expected frequency from the test: \n{Expected_freq}')
```

Statistics value from the test: 10838.372332480214

P_Value of the test is: 0.0

Degree of freedom from the test: 6

****

Expected frequency from the test:

[[221081.86259035 75961.44434981 15290.69305984]

[416408.3330293 143073.60199337 28800.06497733]

[453484.88557396 155812.72247031 31364.39195574]

[385087.91880639 132312.23118651 26633.8500071]]

```
# Assuming significant level is 5%
alpha = 0.05
if p_value < alpha :
  print("We Reject the Null Hypothesis.")
else :
  print("We fail to Reject Null Hypothesis.")
```

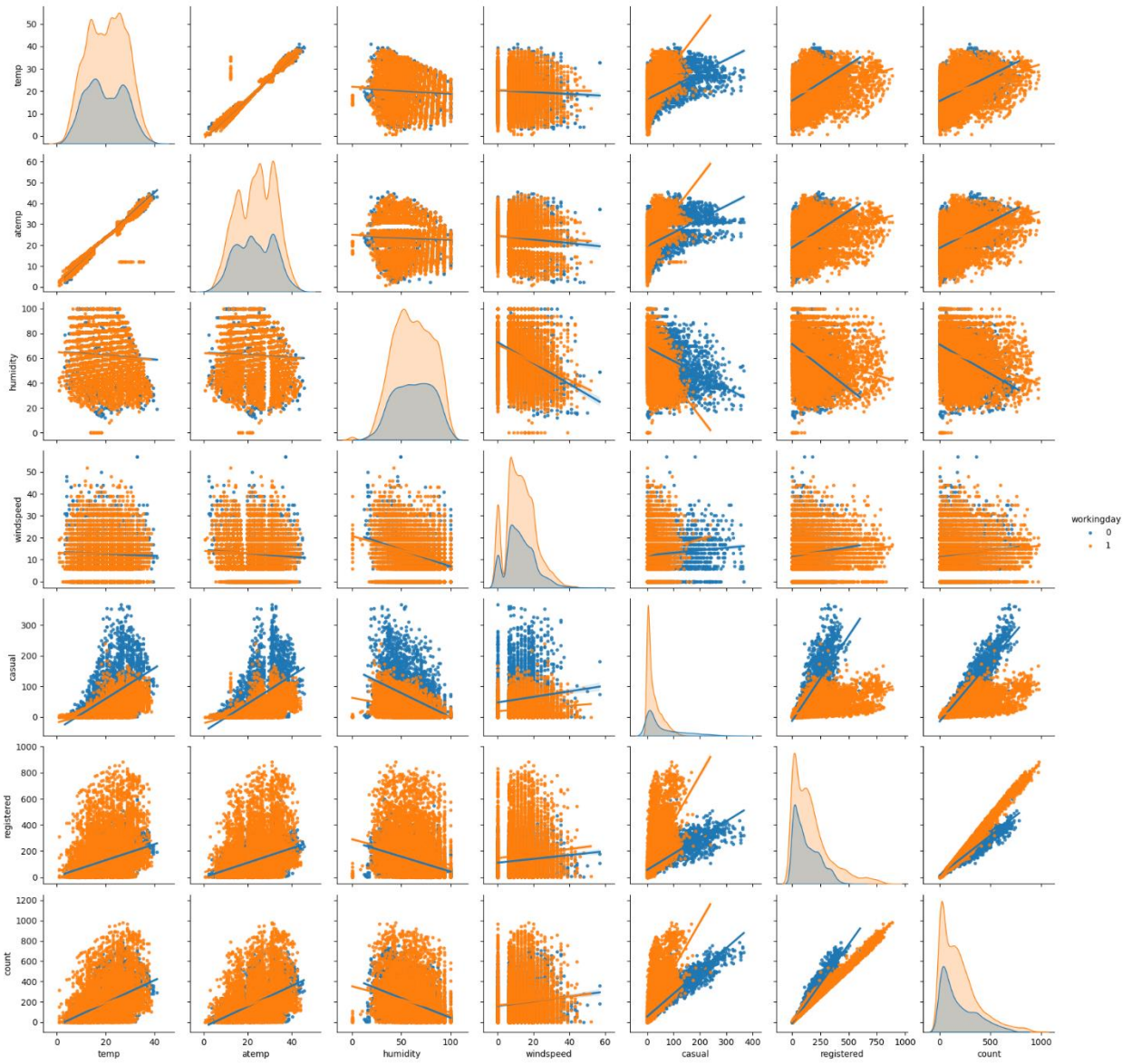
We Reject the Null Hypothesis.

Conclusion of the Test:

Therefore, there is statistically significant dependency of weather and season based on the number of number of bikes rented.

Now will see the pair plot and correlation among all the numeric columns.

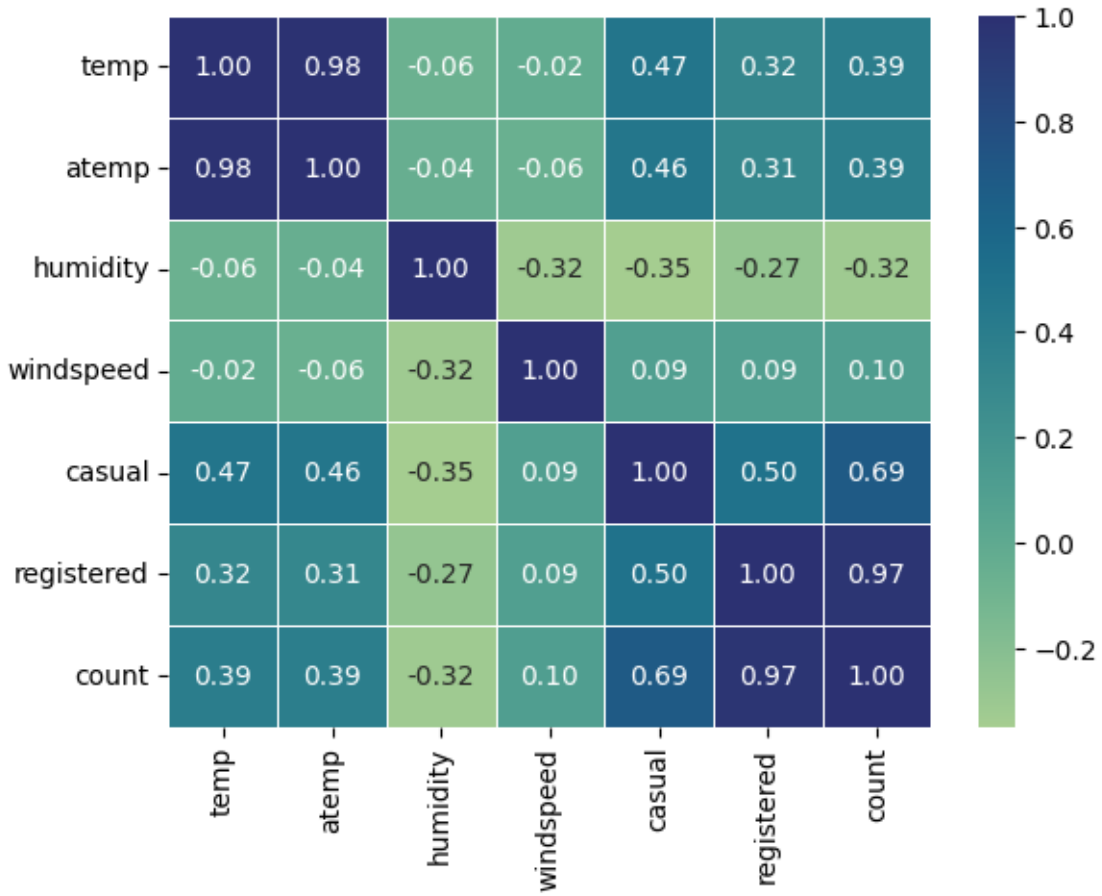
```
sns.pairplot(data = df,
             kind = 'reg',
             hue = 'workingday',
             markers = '.')
plt.show()
```




```
data_corr =
df[['temp','atemp','humidity','windspeed','casual','registered','count']].
corr()
data_corr
```

	temp	atemp	humidity	windspeed	casual	registered	count
temp	1.000000	0.984948	-0.064949	-0.017852	0.467097	0.318571	0.394454
atemp	0.984948	1.000000	-0.043536	-0.057473	0.462067	0.314635	0.389784
humidity	-0.064949	-0.043536	1.000000	-0.318607	-0.348187	-0.265458	-0.317371
windspeed	-0.017852	-0.057473	-0.318607	1.000000	0.092276	0.091052	0.101369
casual	0.467097	0.462067	-0.348187	0.092276	1.000000	0.497250	0.690414
registered	0.318571	0.314635	-0.265458	0.091052	0.497250	1.000000	0.970948
count	0.394454	0.389784	-0.317371	0.101369	0.690414	0.970948	1.000000

```
sns.heatmap(data_corr,annot = True, fmt=".2f", linewidth=.5,cmap="crest")
```



Observations

1. Very High Correlation (> 0.9) exists between columns [atemp, temp] and [count, registered]
2. High positively / negatively correlation (0.7 - 0.9) does not exist between any columns.
3. Moderate positive correlation (0.5 - 0.7) exists between columns [casual, count], [casual, registered].
4. Low Positive correlation (0.3 - 0.5) exists between columns [count, temp], [count, atemp], [casual, atemp]
5. Negligible correlation exists between all other combinations of columns.

Insights

The provided dataset spans from Timestamp('2011-01-01 00:00:00') to Timestamp('2012-12-19 23:00:00'), encompassing a duration of '718 days 23:00:00'.

Among every 100 users, approximately 19 are categorized as casual users, while the remaining 81 are registered users.

The average total hourly bike rental count stands at 144 for the year 2011 and 239 for the year 2012. This indicates an impressive annual growth rate of 65.41% in the demand for electric bikes on an hourly basis.

An intriguing seasonal pattern emerges, showcasing heightened demand during the spring and summer months, followed by a mild decline in the fall and a more pronounced drop in the winter.

Notably, the average hourly bike rental count hits its lowest point in January, closely trailed by February and March.

A distinctive diurnal fluctuation is observed, featuring diminished counts during the early morning hours, a sudden surge in the morning, peak usage during the afternoon, and a gradual tapering off in the evening and nighttime.

It's worth mentioning that over 80% of the recorded time periods experience temperatures below 28 degrees Celsius.

Furthermore, humidity levels surpass 40% for more than 80% of the recorded instances, suggesting that humidity levels predominantly oscillate between the optimal range and overly moist conditions.

In addition, over 85% of the dataset's windspeed measurements fall below 20, indicating generally moderate wind conditions.

Regarding weather conditions, the highest hourly bike rental counts are observed during clear and cloudy weather, followed by misty conditions and rainy weather. Records for extreme weather conditions are notably scarce.

Summary of Hypothesis:

1)Is there any effect of Working Day on the number of electric cycles rented ?

The mean hourly count of the total rental bikes is statistically similar for both working and non-working days.

2)Is the number of cycles rented is similar or different in different weather ?

There is no statistically significant dependency of weather 1, 2, 3 on season based on the average hourly total number of bikes rented.

3)Is the number of cycles rented is similar or different in different season ?

The hourly total number of rental bikes is statistically different for different seasons.

4)Is weather dependent on the season ?

The hourly total number of rental bikes is statistically different for different weathers.

Recommendations

Strategic Seasonal Marketing: Given the evident seasonal pattern in bike rental counts, Yulu can adapt its marketing strategies strategically. Concentrate on promoting bike rentals during the spring and summer seasons when demand peaks. Consider offering seasonal incentives or exclusive packages to entice more customers during these periods.

Dynamic Time-based Pricing: Leverage the hourly fluctuations in bike rental counts throughout the day. Explore the implementation of dynamic time-based pricing, where

rental rates are adjusted to be more affordable during off-peak hours and slightly higher during peak hours. This approach can motivate customers to rent bikes during less congested times, optimizing resource utilization.

Weather-sensitive Promotions: Acknowledge the influence of weather on bike rentals. Create weather-sensitive promotional campaigns targeting customers during clear and partly cloudy conditions, which typically yield the highest rental counts. Yulu can introduce weather-specific discounts to attract more customers during favorable weather conditions.

User-Centric Segmentation: Considering that approximately 81% of users are registered, and the remaining 19% are casual users, Yulu can tailor its marketing and communication strategies with precision. Offer loyalty programs, exclusive incentives, or personalized recommendations to registered users, fostering repeat business. For casual users, emphasize seamless rental experiences and highlight the advantages of bike rentals for occasional use.

Optimized Inventory Management: Analyze demand patterns across different months and fine-tune inventory levels accordingly. During months with lower rental counts such as January, February, and March, Yulu can optimize its inventory to avoid overstocking. Conversely, during peak months, ensure an adequate supply of bikes to meet heightened demand.

Enhanced Weather Data Collection: Given the limited records for extreme weather conditions, consider enhancing data collection procedures for such scenarios. Accumulating more data on extreme weather conditions can facilitate a deeper understanding of customer behavior, enabling adjustments such as offering specialized bike models for different weather conditions or implementing safety measures during extreme weather.

Customer Comfort and Convenience: With generally high humidity levels and temperatures frequently below 28 degrees Celsius, consider enhancing customer comfort. Provide amenities such as umbrellas, rain jackets, or water bottles to elevate the overall biking experience. These thoughtful touches contribute to a positive customer experience and can encourage repeat business.

Collaboration with Weather Services: Explore partnerships with weather services to offer real-time weather updates and forecasts to potential customers. Integrate weather information into marketing campaigns or the rental app to showcase ideal biking conditions, appealing to users who prefer specific weather conditions.

Seasonal Bike Maintenance: Allocate resources for seasonal bike maintenance. Prior to peak seasons, conduct thorough maintenance checks on the bike fleet to ensure optimal performance. Regular inspections and servicing throughout the year can help prevent breakdowns, ensuring customer satisfaction.

Customer Feedback and Reviews: Encourage customers to provide feedback and reviews about their biking experiences. Gathering feedback helps identify areas for improvement, gain insights into customer preferences, and customize services to exceed customer expectations.

Strategic Social Media Marketing: Harness the power of social media platforms to promote Yulu's electric bike rental services strategically. Share captivating visuals of biking experiences in diverse weather conditions, showcase customer testimonials, and engage potential customers through interactive posts and contests. Implement targeted advertising campaigns to reach specific customer segments and boost bookings.

Special Environmental Discounts: Given Yulu's commitment to providing a sustainable solution for vehicular pollution, consider offering special discounts on occasions such as Zero Emissions Day (21st September), Earth Day (22nd April), World Environment Day (5th June), and similar events to attract new environmentally-conscious users.